AUTOMATION OF CLOSED ENVIRONMENTS IN SPACE FOR HUMAN COMFORT AND SAFETY

Report for Academic Year 1991-92

by

Kansas State University

College of Engineering
Departments of Mechanical Engineering, Electrical Engineering, and Chemical Engineering

College of Arts and Sciences
Department of Computer Science

Faculty Advisor: Dr. Allen C. Cogley

June 1, 1992

Submitted to

NASA/USRA
ADVANCED DESIGN PROGRAM

ABSTRACT

This report culminates the work accomplished during a three year design project on the automation of an Environmental Control and Life Support System (ECLSS) suitable for space travel and colonization. The system would provide a comfortable living environment in space that is fully functional with limited human supervision. A completely automated ECLSS would increase astronaut productivity while contributing to their safety and comfort. The first section of this report, Section 1.0, briefly explains the project, its goals, and the scheduling used by the team in meeting these goals. Section 2.0 presents an in-depth look at each of the component subsystems. Each subsection describes the mathematical modeling and computer simulation used to represent that portion of the system. The individual models have been integrated into a complete computer simulation of the CO_2 removal process. In Section 3.0, the two simulation control schemes are described. The classical control approach uses traditional methods to control the mechanical equipment. The expert control system uses fuzzy logic and artificial intelligence to control the system. By integrating the two control systems with the mathematical computer simulation, the effectiveness of the two schemes can be compared. The results are then used as proof of concept in considering new control schemes for the entire ECLSS.Section 4.0 covers the results and trends observed when the model was subjected to different test situations. These results provide insight into the operating procedures of the model and the different control schemes.The appendix, section 5.0, contains summaries of lectures presented during the past year, homework assignments, and the completed source code used for the computer simulation and control system.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

1.1 Project Description

For prolonged missions into space and colonization outside the earth's atmosphere, development of Environmental Control and Life Support Systems (ECLSS) are essential to provide astronauts with habitable environments. ECLS systems for Space Station Freedom (SSF) require semi-autonomous operation to allow environmental control without constant supervision by crew members. The Kansas State University Advanced Design Team is in the process of researching and designing a control system for an ECLSS like that on Space Station Freedom.

The ECLS system for Freedom is composed of six subsystems. The Temperature and Humidity Control (THC) subsystem maintains the cabin temperature and humidity at comfortable levels. The Atmosphere Control and Supply (ACS) subsystem insures proper cabin pressure and partial pressures of oxygen and nitrogen. To protect the space station from fire damage, the Fire Detection and Suppression (FDS) subsystem provides fire sensing alarms and extinguishers. The Waste Management (WM) subsystem compacts solid wastes for return to earth, and collects urine for water recovery. Carbon dioxide and other dangerous contaminants are removed from the air by the Atmosphere Revitalization (AR) subsystem. The Water Recovery and Management (WRM) subsystem collects and filters condensate from the cabin to replenish potable water supplies, and processes urine and other waste waters to replenish hygiene water supplies.

At this time, automation and control of these subsystems have not been fully developed or integrated. A fully integrated and automated ECLS system would increase an astronaut's scientific and observational productivity as well as contribute to their safety and comfort.

1.2 Three Phase Design Schedule

Kansas State University implemented a three phase approach to facilitate the design of a control scheme for an ECLS system. Each phase, consisting of one academic year, represented an evolution and advancement of previous progress.

The first phase consisted of information gathering and determining the particular tasks required for design of the ECLS system. This

accumulated knowledge led to the present organizational structure centered on six interconnected subsystems.

The second phase examined the Air Revitalization subsystem. The concept of a series of mathematical models providing input to a control system was chosen. Prototype models of the CO_2 Removal Assembly governed by a crude expert system controller were developed.

The third phase concentrated on refining the CO_2 Removal Assembly and comparing two control schemes. The two control systems compared are a classical proportional-integral-differential controller and an expert system fuzzy logic controller. The purpose of this study is to enhance the knowledge of these control approaches so choices can be made for the control scheme for the entire ECLS system.

1.3 Third Year Goals

Initially, the proposed goals for the third phase of the design project were to combine the control systems of the six subsystems and form an overall control system for ECLSS with fault diagnosis. However, due to lack of control systems for the individual subsystems, the goals for phase three were reevaluated.

The overall objective of the final year is to develop and compare expert and classical systems of control on a computer simulation of the CO_2 Removal Assembly of the ECLSS.

Goals for reaching the final objective begin with creating a mathematical model and a computer simulation of the CO_2 Removal Assembly. Concurrently, development of the classical and expert systems of control were performed. The next goal is to integrate the control systems and the computer simulation together and evaluate and compare the effectiveness of each control system. The comparison will be used as a proof of concept to evaluate the use of expert systems to control the entire ECLSS.

A list of the goals for the third and final year are as follows:

1. Complete the computer simulation of the CO_2 Removal Assembly.
2. Create a set of rules for the expert control system of the CO_2 Removal Assembly.
3. Create a classical controls system for the CO_2 Removal Assembly.

4. Establish a means of communication between the mathematical model and the two controls systems.

5. Analyze the dynamic response of the simulation and compare the two methods of control.

1.4 Academic Year Time Table

The year started with an introduction to the advanced design teams objectives for the project. Several lectures given by faculty and graduate members of Kansas State University introduced the design team to mathematical modeling, simulation and control. This introduction lasted until September 25th.

The next step was to plan objectives for the first semester, and decide what should be accomplished for the third year. A comparison of expert system controls and classical system controls for a subassembly of the ECLSS was decided upon as the third year main objective.

Three modeling groups and two controls groups were formed to develop models for the individual parts of the CO_2 Removal Assembly. The modeling of the desiccant beds, the blower/ precooler, and the CO_2 sorbent beds began about October 23rd, with completion deadlines planned for November 10th. These models were to be integrated together forming a computer simulation of the overall process. A presentation of the progress was given on November 25th.

Documentation of the semesters work started on December 2nd and a semester report was submitted to the faculty advisors on January 23rd.

The final semester goals were to refine the math models formulated during the fall semester, complete implementation of controls on the CO_2 Removal System, and create a user interface using X Windows.

On January 30th two modeling groups were formed. One to refine the math model of the sorbent beds and another to find information on the inputs and outputs to the CO_2 Removal Assembly. Two other groups were formed to implement classical and expert controls on the CO_2 Removal Assembly. On March 4th, work on a Graphical User Interface (GUI) using X Windows was begun. All phases were completed for the All University Open House on April 4th with displays in the College Engineering and the College on Arts & Sciences.

Documentation and a presentation for the USRA/NASA Design Program was begun on April 6th and continued until the day of the conference on June 15th.

1.5 Design Team Description

The Advanced Design Team at Kansas State University is composed of students from several academic disciplines. Currently participating disciplines include computer science, and mechanical engineering and chemical engineering. The team's Graduate Teaching Assistant is an electrical engineer. Plans are under way to recruit students from electrical and computer engineering for the final semester. Faculty support comes from the mechanical, electrical, chemical, and computer engineering departments as well as the computer science department.

2.0 MATHEMATICAL MODELING

2.1 CO_2 Removal Assembly

2.1.1 Introduction

The Carbon Dioxide Removal Assembly, designed to remove carbon dioxide from the cabin air, involves removal of CO_2 by molecular sieves. The process is required to remove carbon dioxide generated by the respiratory processes of the astronauts and to maintain acceptable levels of carbon dioxide within the cabin.

Figure 2.1-1 is a block diagram representation of the CO_2 Removal Assembly. The system takes input air from the Temperature Humidity Control Subsystem (1), and valves (2,11) direct the air flow allowing it to flow across one of the desiccant beds (3,10) which dehumidify the air using Zeolite 13X and Silica Gel. The moisture must be removed to avoid poisoning the desiccant found in the adsorbing sorbent bed (8,14). Because the dry air is heated in the process, it is forced across a heat exchanger (6) by a blower (5), and the air is cooled before being sent through a sorbent bed. The sorbent beds remove the carbon dioxide by means of Zeolite 5A, which acts as a molecular sieve adsorbing the carbon dioxide. The dry air returning from the molecular sieves through unidirectional control valves (13,9) is revitalized by the moist desiccant of the second desiccant bed (10). After the air is rehydrated it is then returned to the Temperature and Humidity Control Subsystem (12) and redistributed throughout the cabin.

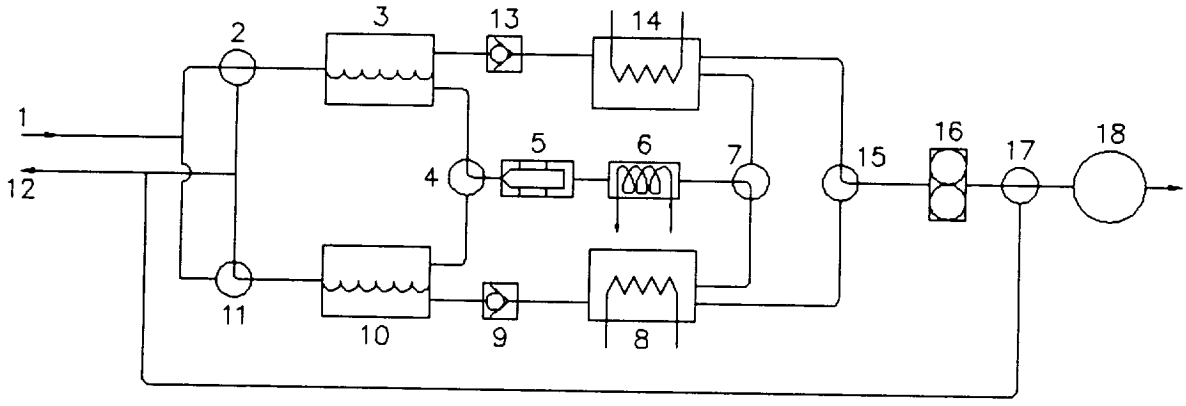

Figure 2.1-1: CO_2 Removal Assembly

Concurrently, a second desorbing sorbent bed (14) is being heated causing the separation of the carbon dioxide from the desiccant. The desorbed carbon dioxide is drawn from the bed by means of a pump (16) and is sent to an accumulator tank (18). After the adsorbing desiccants have become saturated, the desorbing beds are once again dry. The control valves (5,7,15) redirect air flow in the system. The previously adsorbing beds begin the desorbing process and the previously desorbing beds begin adsorbing. The system is presently configured to cycle every thirty minutes.

Mathematical models of the various components were created to allow analysis of the subassembly's performance. The role of the modeling is to duplicate the actual systems response to a given set of parameters. Knowing how an actual system should respond, it is possible to explore control systems for use in governing the subassembly. The control systems regulate the state variables throughout the subassembly.

2.1.2 Desiccant Beds

DESCRIPTION

The purpose of the desiccant beds is to remove water vapor from the incoming air. This function is necessary because water will poison the zeolite used in the CO_2 adsorption process. Water vapor removal is achieved by means of two desiccants, Silica gel and Zeolite 13X. High-humidity air coming into the CO_2 Removal Assembly flows first over the Silica gel, which removes some of the water and brings the relative humidity down to a low level. Since Zeolite 13X works well at low relative humidities, it is then used to remove most of the remaining water vapor before the air is sent on to the blower and precooler. The exiting air is not only dry, but also heated from the release of energy required to condense the water vapor. As adsorption is taking place in one bed, the other desiccant bed is rehumidifying the air returned from the CO_2 sorbent beds through a desorption process. The desorbing cycle is just the reverse of adsorbing in that hot, dry air flows across the Zeolite 13X first, then over the Silica gel, and is cooled and humidified in the process.

MATH MODEL

Assumptions

1. Equilibrium relative humidity is a linear function of load for Silica gel, while that for Zeolite 13X can be approximated with two straight lines.

2. All heat transfer is between the air and water only.

3. Water in the desiccant is evenly distributed at all times.

4. Beds remain at room temperature.

5. Air is heated before desorption.

6. The water is removed and released at constant pressure.

7. The specific heat of the air is constant.

8. Enthalpy of condensation and vapor saturation pressure are accurately represented by linear and fourth order least squares curve fits, respectively.

9. Equilibrium relative humidity, which is a function only of the load on the desiccant bed, is achieved.

Equations

A thermodynamic analysis of the air flowing through the bed yields the following equations used for the mathematical model. Equation 1 is a curve fit for vapor saturation pressure as a function of saturation temperature. Values for the curve fit were obtained from the steam tables in an appendix of Thermodynamics, An Engineering Approach by Cengel and Boles. The expression is

$$P_{sat}=.3972+.0629T+.001099T^2+1.705x10^{-5}T^3+6.192x10^{-7}T^4. \quad (1)$$

From the same text, relationships equating pressures, mass, and relative and absolute humidities are shown in equations 2, 3, and 4 as

$$P_v=\phi P_{sat}, \quad (2)$$

$$\omega=\frac{.622P_v}{P-P_v}, \quad (3)$$

$$m_v=\frac{\omega m_f}{1+\omega}. \quad (4)$$

The law of mass conservation can be applied to the model when evaluating the air inside the desiccant bed. The mass of dry air in the bed (m_a) equals the total mass of air in the bed (m_f) minus the mass of water vapor in the air (m_v) state as

$$m_a=m_f-m_v. \quad (5)$$

The load on the desiccant can be defined as the mass of water vapor absorbed versus the mass of desiccant in the bed expressed as

$$L=\frac{m_{ads}}{m_{tank}}. \tag{6}$$

Equilibrium load curves were provided by Dr. Byron Jones of Kansas State University from an ASHRAE reference. Data points taken from these curves of the load versus relative humidity for the desiccants were curve fit using a least squares method fortran program written by Dr. Kirby Chapman, professor of Mechanical Engineering at Kansas State University. The resulting linear curve fits are given in Equations 7, 8, and 9. The fit for Zeolite 13X was approximated using the two straight lines of equations 8 and 9. The results are

$$\phi=\frac{L}{.5263} \quad (Silica\ gel), \tag{7}$$

$$\phi=.4L \quad (L\leq.17,\ Zeolite\ 13X), \tag{8}$$

$$\phi=.068+40(L-.17) \quad (L>.17,\ Zeolite\ 13X). \tag{9}$$

The mass of water removed from the air can be determined using the thermodynamic relationship

$$m_r=m_v-\omega m_a. \tag{10}$$

Enthalpy, the incoming air temperature, and the outgoing air temperature are related by the thermodynamic reations

$$h_{fg}=2502-2.389T_{in}, \tag{11}$$

and

$$T_2=T_{in}+\frac{m_r h_{fg}}{(m_f-m_r)c_p}. \tag{12}$$

The remaining equations are simply relationships for the rates of mass absorbed or desorbed, and the change in mass of air vapor and air in the bed versus time. These equations govern the mass transfer of the water from the desiccant and the air. Utilized in a finite time step process, the following expressions determine the success of the desiccants in removing water from the air.

$$\frac{dm_{ads}}{dt} = \frac{dm_r}{dt}, \tag{13}$$

$$\frac{dm_v}{dt} = -\frac{dm_r}{dt}, \tag{14}$$

$$\frac{dm_f}{dt} = -\frac{dm_r}{dt}. \tag{15}$$

The symbols used in the mathematical model are defined as follows.

P_{sat} = vapor saturation pressure (kPa)
T = temperature of air being evaluated (°C)
P_v = partial pressure of the water vapor (kPa)
ϕ = relative humidity
ω = absolute humidity
P = total pressure of the air (kPa)
m_v = mass of vapor in the air (kg)
m_f = total mass of air in the bed (kg)
m_a = mass of dry air in the bed (kg)
L = load on the desiccant
m_{ads} = mass of water adsorbed by the desiccant (kg)
m_{tank} = mass of desiccant material (kg)
m_r = mass of water removed from the air (kg)
h_{fg} = enthalpy of condensation (kJ/kg)
T_{in} = temperature of incoming air (K)
T_2 = temperature of outgoing air (K)
c_p = specific heat of air (kJ/kg·K)
$\frac{dm_{ads}}{dt}$ = rate at which the desiccant adsorbs water (kg/s)
$\frac{dm_r}{dt}$ = rate at which water is removed from the air (kg/s)
$\frac{dm_v}{dt}$ = change in mass of vapor in the air (kg)
$\frac{dm_f}{dt}$ = change in total mass of air in the bed (kg)

MODELING TECHNIQUES

A computer program was written to simulate the performance of the desiccant beds in time. This was accomplished by choosing a small time step, and then evaluating the above equations for the mass in the bed during that time step. The first calculations on this mass

determine the composition of the air based on the input conditions. Next, the equilibrium relative humidity is found from the load on the first desiccant to come in contact with the air flow -- Silica gel for adsorbing, Zeolite 13X if desorbing. The change in the amount of water in the air is found from the difference between the input and equilibrium states, with removal of water from the air considered positive. The temperature change of the air is dependent on the amount of water removed. However, changing the temperature of the air also alters its relative humidity (but not the absolute humidity) so that it no longer matches equilibrium. Therefore, an iterative procedure is required to find the point where the output temperature and its corresponding humidities are consistent with the equilibrium relative humidity and the new amount of water in the air. Execution of the program showed that an average of five iterations were needed. With this done, the mass of air is sent on to the other desiccant, and the calculations are repeated to produce the final output conditions of the air from the bed.

It should be noted that little distinction is made between the adsorbing and desorbing cycles. This is because the desiccant itself is unaware of the intended function; it merely reaches equilibrium with the conditions it is given. Naturally, adsorption will occur when cool, humid air passes over desiccant with a low loading. In order for the loaded desiccant to be desorbed by the dry air on the return trip, the air must first be heated by means of a heat exchanger because hotter air holds more water vapor. Since the current system does not account for heating the air, the program sets the input temperature to a sufficiently high value on the desorption cycle.

RESULTS

Many simulations were run with the program, each time varying one parameter to observe its effect. Since the temperature and relative humidity of the air being sent to the CO_2 desorption process (and then back to the cabin) are the most important, it is no surprise that they have the greatest effect on the performance of the desiccant beds. Temperature is the most influential of all parameters because relative humidity is a function of temperature. The mass of the desiccant is also an important parameter, since it affects the loading. Ideally, any problems with the amount of absorbed water vapor could be solved by varying the desiccant mass within the bed.

The following are expected normal operating conditions, inputs, and experimentally determined values:

Mass flow rate of air	.2 kg/s
Adsorption input temperature	300 K

Desorption input temperature	363 K
Adsorption input relative humidity	100%
Total pressure of the air	101.325 kPa
Specific heat of air	1.006 kJ/kg·K
Initial mass of water adsorbed	0 kg
Mass of each desiccant	40 kg.

Figures 2.1-2 and 2.1-3 show the behavior of the desiccant beds from startup under standard conditions. One initial condition of the process assumes the desiccants to be completely dry. Desiccants have a tendency to retain some moisture which will not be retrievable during the desorption cycle. Over time, the amount of residual water will asymptotically approach an equilibrium value which will represent the maximum amount of irretrievable water vapor retained by the desiccant. This characteristic accounts

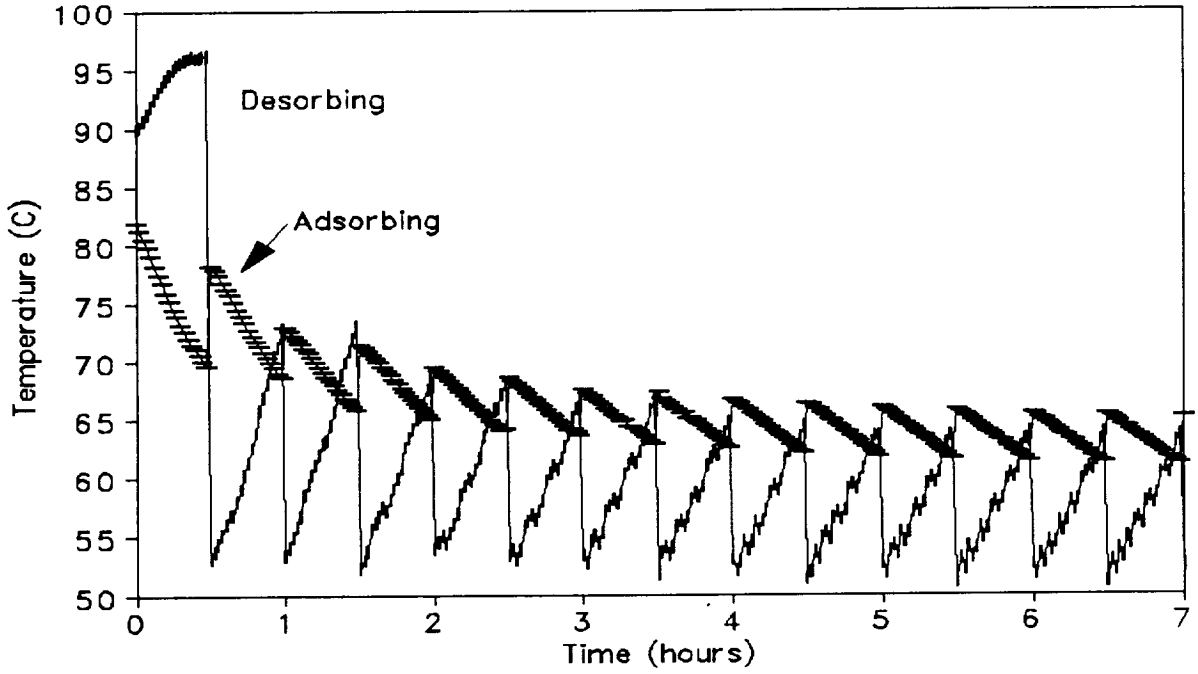

Figure 2.1-2: Standard Temperatures Leaving Desiccant Bed

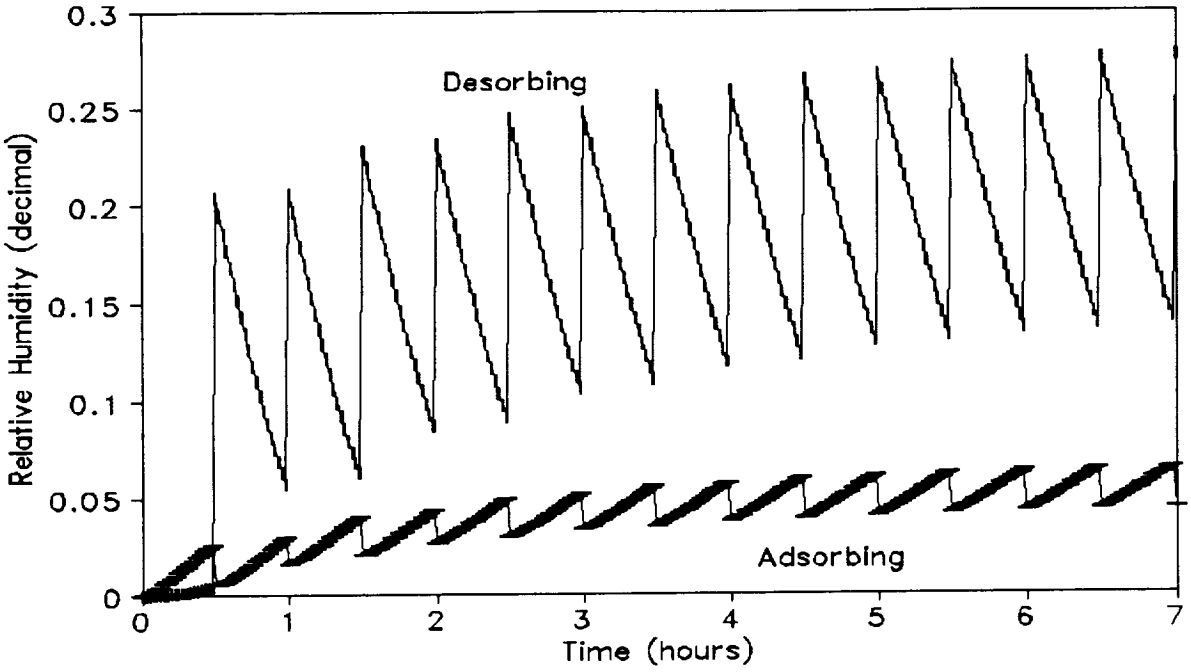

Figure 2.1-3: Standard Humidities Leaving Desiccant Bed

for the transient responses of the adsorbing and desorbing cycles in both figures. Notice from the figures that after startup, the output temperatures and humidities reach a steady pattern. After reaching a steady state, the average outputs of the adsorption cycle tends to be a temperature of about 63°C with roughly 6% relative humidity, while the desorption cycle is returning air at averages of about 58°C and 20% relative humidity.

Because heat is generated when water vapor is adsorbed by desiccant, the amount of water vapor adsorbed is directly related to the output temperatures from the beds. In addition, the amount of water vapor adsorbed by a desiccant during a thirty minute cycle may be represented by a decaying exponential function. Examining the first 30 minute adsorbing cycle of Figure 2.1-2, one may notice a falling output temperature due to the decrease in heat generated by the decaying amount of water vapor adsorbed during each time step.

Since the adsorption and desorption cycles are essentially reciprocal, the exponential decay of water vapor released during the desorption cycle, and the increase in output temperature can also be explained. The desorption cycles are conducted with the air flow being at a higher temperature. That fact coupled with the removal of CO_2 result in the differing magnitudes of change in relative humidity.

2.1.3 Blower and Precooler

DESCRIPTION

The blower/precooler is the second process of the CO_2 Removal Assembly. This process utilizes a variable speed blower to force cabin air through the CO_2 Removal Assembly and a crossflow heat exchanger to cool air received from the desiccant beds. Air leaving the precooler is then directed on to the sorbent beds where carbon dioxide is removed.

MATH MODEL

Assumptions

1. Pressure drop across the cooler is negligible.

2. Water specific heat and air density are constant properties.

Specification of Heat Exchanger

1. Heat exchanger effectiveness is 0.80.

2. Heat exchanger coolant is water.

3. Mass flow of coolant is 3.79 kg/min (500 lbm/hr.).

4. Inlet temperature of coolant is 15°C (59°F).

5. Cross-sectional area is 11.1 m^2 (120 FT^2).

Equations

Equation 1 is the maximum heat transfer rate that can be drawn from the air by the heat exchanger. Equation 2 is the actual heat transfer rate using the heat exchanger effectiveness. The concepts give

$$\underline{Q}_{max}(\underline{t}) = \underline{C}_{p,air}\underline{M}_{air}(\underline{T}_{h,i}(\underline{t}) - \underline{T}_{c,i}), \quad (1)$$

$$\underline{Q}_{act}(\underline{t}) = \epsilon \underline{Q}_{max}(\underline{t}). \quad (2)$$

Equation 3 is the temperature of the air leaving the precooler as a function of time given the inlet temperature, the actual heat transfer rate and the mass and specific heat of air. Symbolically this can be written as

$$\underline{T}_{h,out}(\underline{t}) = \underline{T}_{h,i}(\underline{t}) - \frac{\underline{Q}_{act}(\underline{t})}{\underline{C}_{p,air}\underline{M}_{air}}. \quad (3)$$

The inlet temperature function used to test the program is given as

$$\underline{T}_{h,i}(\underline{t}) = 2\underline{t} + \underline{T}_o. \quad (4)$$

This function varies with time because the outlet temperature from the desiccant beds is not constant. Note that the equation used does not represent the actual desiccant bed outlet temperature, but is only an approximate linear function used to test the response of the precooler model at extreme conditions.

The symbols used in the above equations are defined as follows:

$C_{p,air}$ = specific heat of air (kJ/kg•K),
ϵ = heat exchanger effectiveness,
M_{air} = mass flow rate of air into cooler (kg/s),
$Q_{act}(t)$ = actual heat transfer between two fluids (kW),
$Q_{max}(t)$ = maximum amount of heat transfer between two fluids, (hot inlet air and cold coolant water) (kW),
$T_{c,i}$ = inlet temperature of cooling water (K),
$T_{h,i}(t)$ = inlet temperature of hot air (K),
$T_{h,out}(t)$ = outlet temperature of cooled air (K),
T_o = initial temperature of inlet air (300 K),
t = time (minutes).

RESULTS

Figure 2.1-4 compares the inlet and outlet temperatures of the precooler. As shown in the figure, the inlet temperature of the air increased from 300 K to 420 K over 60 minutes while the precooler outlet temperature varied between 290 K and 313 K. This precooler temperature range was used for sorbent bed inlet air temperatures in other modeling for that component.

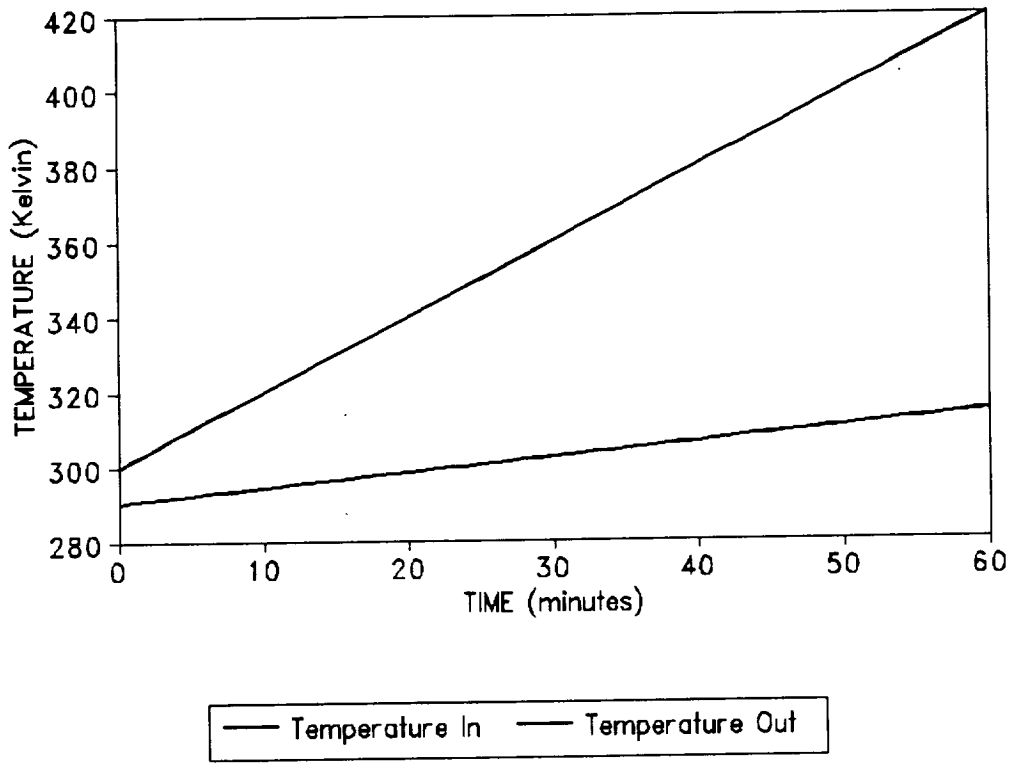

Figure 2.1-4: Precooler Inlet and Outlet Temperatures

2.1.4 CO_2 Sorbent Beds

DESCRIPTION

The CO_2 Sorbent Beds are the third step of the CO_2 removal process. Their purpose is to separate CO_2 from the air, return the air to the desiccant beds and send the removed CO_2 gas to an accumulator tank. This is done using two adsorption beds containing Zeolite 13X.

MATH MODEL

Assumptions

1. The exiting air temperature equals the temperature of CO_2 in the bed.

2. The heat transfer coefficient of Zeolite 5A is a linear function of temperature.

3. Power supplied to the beds is either on (1000 J/s), off (0 J/s), or being removed at 1000 J/s.

4. Thermal equilibrium for the sorbent beds negates dependence on bed length.

5. Assume that a four man loading supplies CO_2 at an average rate of 5.046×10^{-4} kg/s (this can be a function of time).

Equations

The mass of CO_2 contained within an adsorbing bed at a time T is equal to the mass adsorbed at some time T-t plus the mass transferred during time t. The result is of the form given in Equation 1 below. This mass transfer is brought about by the sorbent bed removing the CO_2 based on a difference in the equilibrium and actual partial pressures. This mass transfer is given by equation 2 shown below. The expressions are

$$m_T = \dot{m}_{T-t} + \dot{m}_t * \Delta t, \tag{1}$$

and

$$\dot{m}_t = \frac{(P_{CO_2} - P_{equilibrium})\, V_{tank}}{R_{CO_2} T_{tank}}. \tag{2}$$

The ideal gas law applied in the above equations provide a simple relationship between the mass flow rates that are desired and the partial pressures which are known.

The energy involved in the mass transfer and accompanying phase change results in the bed temperature being raised. In addition, during the desorbing phase the bed is heated to drive off the CO_2 and this results in a further increase in bed temperature. This temperature change is governed by the following energy balance:

$$T_{bed_{new}} = T_{bed_{old}} + \frac{HEAT}{m_{air} CV_{air} + m_{CO_2} CV_{CO_2} + (m_{bed} + m_{abs})\, CV_{bed}}. \tag{3}$$

Finally, the equilibrium partial pressure curve was derived from curve fitting data provided from Marshall Space Flight Center.

This data shows that the equilibrium partial pressure of the bed and air flow is a function of both the corresponding temperatures and most importantly the bed loading, or current level of absorbed mass. This information is expressed as

$$PPCO_2 = .1333 \exp\left(10^{load} - 3.065 + .02622\, T_{bed} - 4.684E\text{-}5 * T_{bed}^2\right). \quad (4)$$

The symbols used in the above equations are defined as follows:

T_{bed} = temperature of sorbent bed (K),
m_{CO2} = mass of CO_2 transferred (kg),
P_{CO2} = partial pressure of CO_2 (kPa),
P_{equil} = equilibrium CO_2 concentration of bed (kPa),
R_{CO2} = CO_2 gas constant (kPa m^3)/(kg K),
V_{bed} = volume of sorbent bed (m^3),
HEAT = power applied to bed (J/s),
m_{air} = mass of air in bed (kg),
m_{bed} = mass of sorbent bed (kg),
m_{abs} = mass of CO_2 absorbed in bed,
cv_{air} = specific heat of air (J/kg K),
cv_{CO2} = specific heat of CO_2 (J/kg K),
cv_{bed} = specific heat of sorbent bed (J/kg K).

RESULTS

The system is designed to include two sorbent beds which alternate between the adsorbing and desorbing roles. While one bed is adsorbing the CO_2 flowing through the system, the other is being heated and its previously adsorbed CO_2 is released and pumped out to the accumulator tank. The Figure 2.1-5 shows the loading curves for the two beds. The increasing curve is indicative of the bed that is loading, while the decreasing bed's loading is shown as the curve that is falling off.

The work done on the subroutine involved a total overhaul of the previous semester's model due to unacceptable limitations in the earlier version's performance. This work included enhancing the accuracy of the model's portrayal of the actual phenomena, and increasing the subroutine's compatibility with the main program.

After the fundamental flaws were corrected, the problem of fine tuning the desorption process was examined. Two major criteria were established as defining the problem. First it was neccessary to desorb all the CO_2 in the half hour cycle, and second, it was neccessary to provide almost pure CO_2 gas to the accumulator tank which feeds the Bosch reactor.

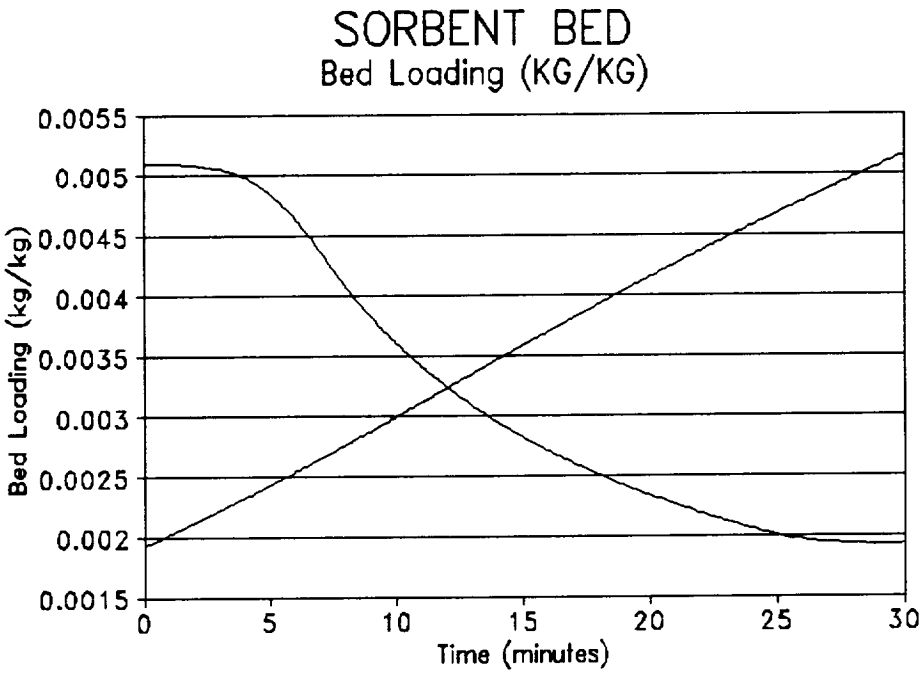

Figure 2.1-5 CO_2 Bed Loads

The first problem was solved by incorporating a heating/cooling jacket to the sorbent bed. This allowed the temperature of the bed to be raised which resulted in a lower affinity for the adsorbed CO_2. The immediate problem with this was the need to cool the bed before returning it to the adsorbing cycle. A 1000 watt heating/cooling jacket was found to be adequate to accomplish both of these ends.

The result of this bed heating was that the CO_2 gas was desorbed into the bed to mix with the residual air still in the tank after the cycle switch. During the first three minutes of the adsorbing cycle the bed is vented into the Temperature and Humidity Control assembly to avoid contamination of the input CO_2 gas for the Bosch Reactor. The final result of the heating and cooling curves can be seen in Figure 2.1-6.

The adsorbing process is not unidirectional in that the bed achieves an equilibrium not neccessarily in phase with the desired result. The temperature dependency of the equilibrium partial pressure results in a transient desorption phase in the beginning of the adsorption cycle. The resulting change in equilibrium

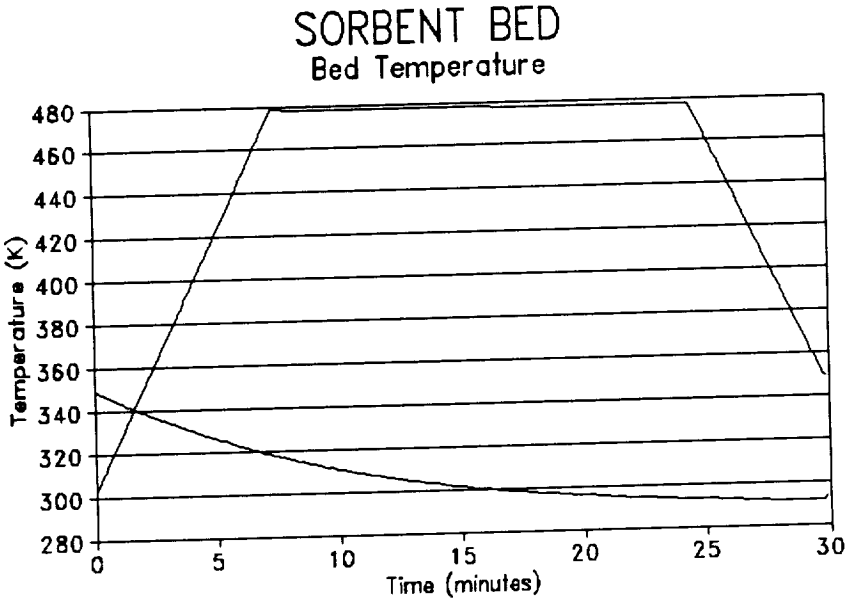

Figure 2.1-6 CO_2 Bed Temperatures

partial pressure results in the mass transfer of CO_2 as the bed strives to maintain equilibrium with the ever emptying chamber. Figure 2.1-7 shows the partial pressure of CO_2 in the chamber as the bed desorbs and the pump removes CO_2. As the bed heats up the equilibrium partial pressure is increased, hence the rise in the graph. However as the bed desorbs its CO_2 its equilibrium partial pressure falls off. Finally the cooling jacket begins to lower the bed temperature and the partial pressure begins to lower even further. The end result is that as the bed returns to the adsorbing cycle, its equilibrium partial pressure is very low and it is able to immediately begin adsorbing CO_2.

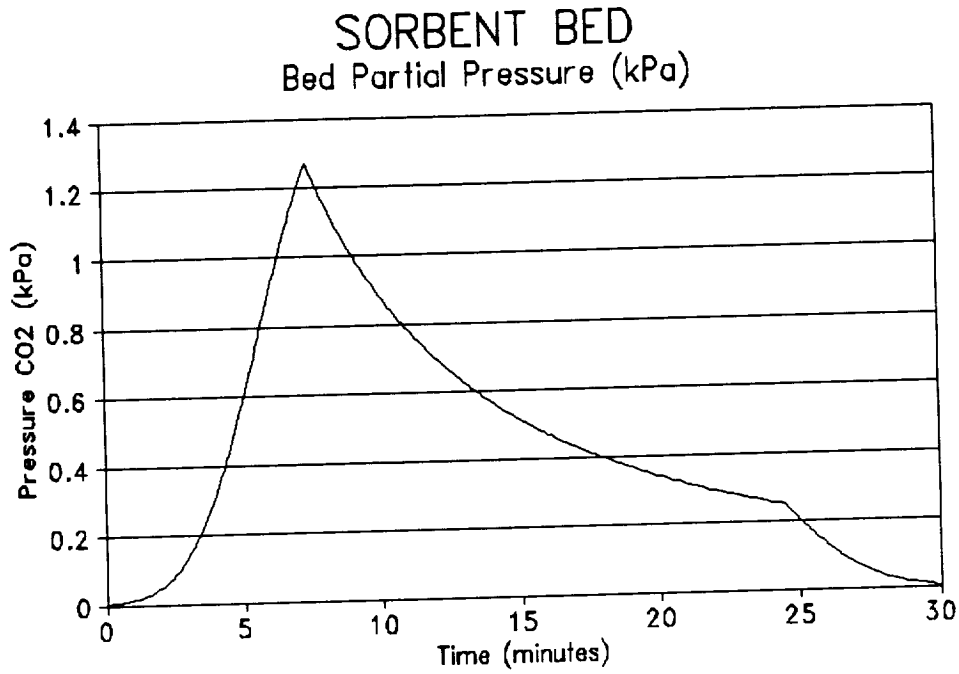

Figure 2.1-7 CO_2 Pressure for Adsorbtion Cycle

2.1.5 Pump and Accumulator

DESCRIPTION

The CO_2 pump and CO_2 accumulator tank is the final process of the CO_2 Removal Assembly. CO_2 adsorbed by the sorbent beds is released and pumped into an accumulator tank. The CO_2 accumulator tank stores the pumped CO_2 until it is needed by the CO_2 Reduction Assembly.

MATH MODEL

Assumptions

1. The CO_2 pump is a fixed displacement, rotary vane pump.
2. Pump is 100% efficient.

3. Pump operates under adiabatic and isentropic conditions.

4. Accumulator tank is perfectly insulated.

5. CO_2 is an ideal gas.

6. CO_2 specific heat is constant.

Equations

The rate of CO_2 mass pumped to the accumulator tank (m_p) can be determined by the following equation that used the ideal gas law: PV=mRT. Using the pump speed (sp=rev/sec.) and the volume displaced per revolution ($V=m^3/rev$) one can obtain the rate equation as

$$\dot{m}=\frac{P\ sp\ V}{R\ T}. \qquad (1)$$

The temperature on the outlet side of the pump is calculated using the pressure differences on each side of the pump and the assumption that the inlet pressure to the pump is constant. This gives

$$T_{po}=T_{pi}\left(\frac{P_T}{P_{pi}}\right)^{(1-1/k)}. \qquad (2)$$

The time change in enthalpy at both the inlet and outlet of the tank is given by

$$H_{in}=m_p((T_{po}-273)C_p+h_0), \qquad (3)$$

and

$$H_{out}=m_o((T_T-273)C_p+h_0). \qquad (4)$$

The initial mass and internal energy conditions of the tank are determined knowing the initial tank temperature, pressure, and tank volume. The relationships are given by

$$m_{Ti}=V_T P_{Ti}/(RT_{Ti}), \qquad (5)$$

and

$$U_{Ti}=(T_{Ti}-273)C_v m_{Ti}. \qquad (6)$$

The overall mass within the tank as a function of time is found by subtracting the CO_2 drawn from the tank from the mass pumped into the tank and adding it to initial CO_2. This can be stated as

$$m_T = m_{Ti} + (m_p - m_o)\, t_{stp}. \quad (7)$$

The current tank temperature and pressure is found using the first law of thermodynamics and the ideal gas law respectively. These concepts give

$$T_T = T_T + m_{CO_2,i} C_{v,CO_2} T_i / (m_{CO_2,i} + m_T), \quad (8)$$

and

$$P_T = m_T R_{CO_2} T_T / V_T. \quad (9)$$

The symbols used in the mathematical model are defined as follows:

m_p = mass flow rate of CO_2 from the pump (kg/sec),
sp = speed of the pump (revolutions/sec),
V_p = volume displaced by the pump per revolution (m^3),
P_{pi} = inlet pressure to pump (kPa),
R = ideal gas constant (kPa*m^3/kg*K),
T_{po} = outlet temperature of the pump (K),
T_{pi} = inlet temperature of the pump (K),
P_T = pressure of CO_2 in the tank (kPa),
k = specific heat ratio of CO_2,
H_{in} = enthalpy of inlet CO_2 stream to the accumulator (J/s),
H_0 = enthalpy of CO_2 at the reference temperature (J/kg),
C_p = constant pressure heat capacity of CO_2 gas (J/kg*K),
H_{out} = enthalpy of outlet CO_2 stream from accumulator (J/s),
m_o = mass flow rate of CO_2 leaving accumulator (kg/sec),
m_{Ti} = initial mass of CO_2 in accumulator tank (kg),
V_T = volume of accumulator tank (m^3),
P_{Ti} = initial pressure inside tank (kPa),
T_{Ti} = initial temperature in tank (K),
U_{Ti} = initial internal energy of tank (J),
m_T = current mass of CO_2 in the accumulator tank (kg),
t_{stp} = time elapsed between calculations (sec),
U_T = current internal energy of tank (J),
T_T = current temperature of tank (K),
C_v = constant volume heat capacity of CO_2 (J/kg*K),
P_T = current pressure inside tank (kPa).

RESULTS

The development of a pump and accumulator simulation was assigned

as homework to the design team as an introduction to math modeling. The homework summary and source code can be found in the appendix under section 5.4 Homework 1 Summary as well as results for the mathematical model. The graph of conditions is in Figure 2.1-8

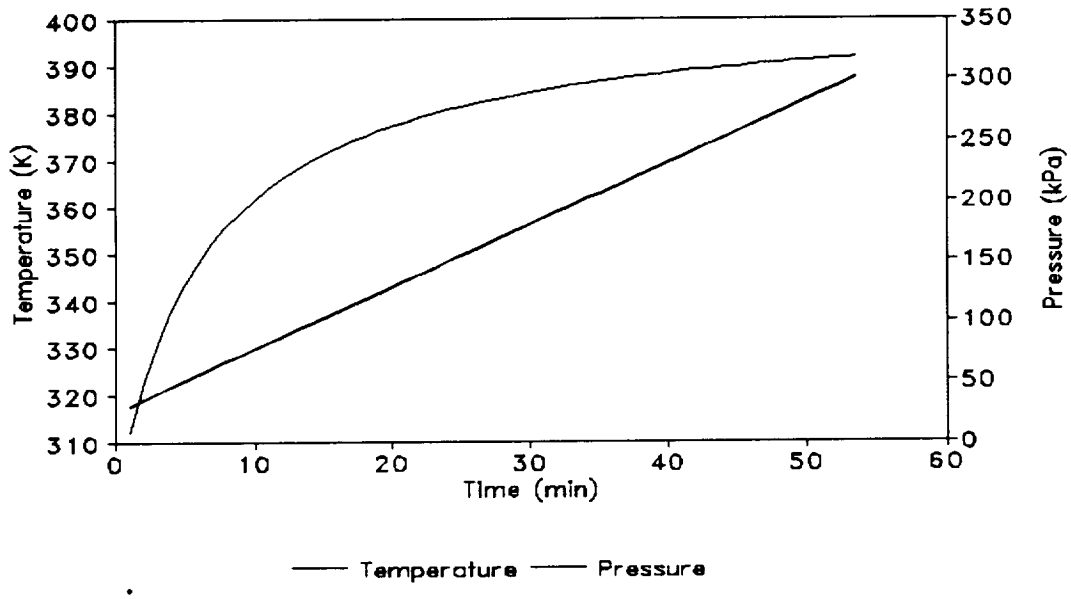

Figure 2.1-8: Homework 1 Case 1 Data

The linear curve corresponds to tank pressure, and is listed against the right hand axis. The upper curve is the tank pressure and is read off the left axis. The results show tank pressure approaching a maximum value while the pressure continues to increase.

2.2 CO_2 Reduction Assembly

INTRODUCTION

The following description is used to define the input variables to the CO_2 removal model. The CO_2 Reduction Assembly consists of a Bosch reactor, heater, compressor, condensing heat exchanger and dynamic separator, and recuperative heat exchanger. A schematic of the CO_2 Reduction Assembly is given in figure 2.2-1.

DESCRIPTION

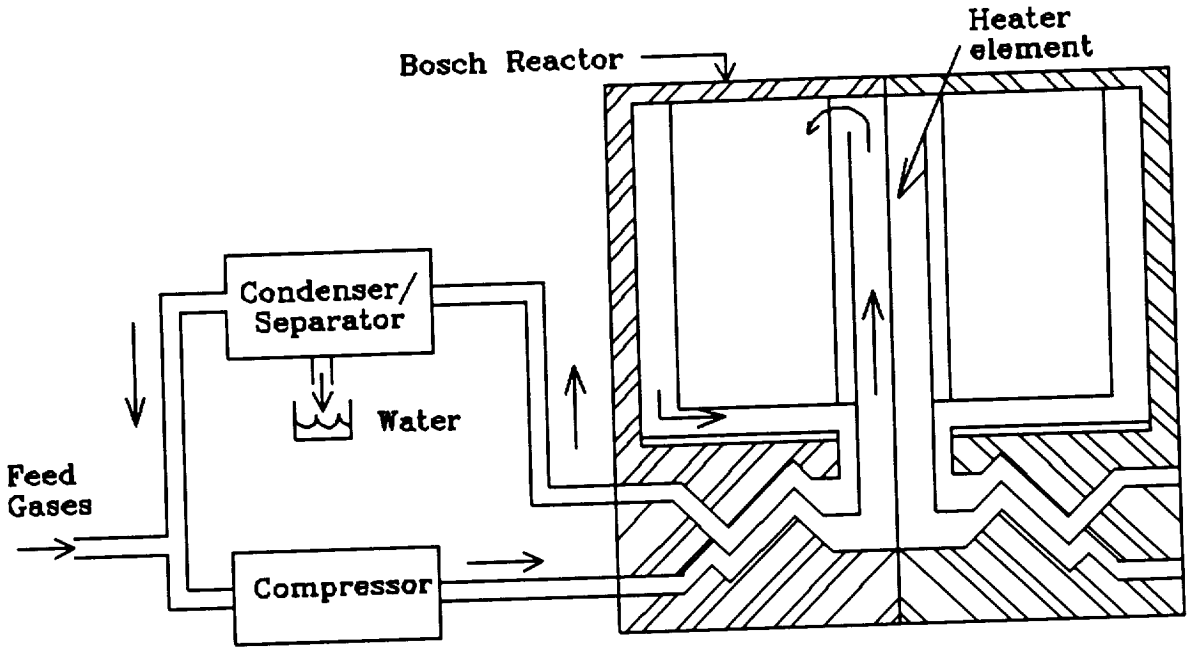

Figure 2.2-1: CO_2 Reduction Assembly

The CO_2 flows from the CO_2 accumulator tank through a pressure control valve to the Bosch Reactor. The CO_2 is mixed with an excess of Hydrogen from the Oxygen Generation Assembly. The two react to give solid carbon, which is collected in the Carbon Filtration Cartridge, and water. The reaction is less than 10 % efficient so the remaining reactants are recycled. After passing through a condensing heat exchanger and dynamic separator, the dried reactants are mixed with incoming "makeup" reactants. This mixture is put into the compressor and cycles through the Bosch Reactor again. This way 99% of the CO_2 is reduced to solid Carbon and water.

The reaction that takes place within the reactor is:

$$CO_2 + 2H_2 = 2H_2O + C + HEAT \qquad (1)$$

The heat generated by the reaction lowers the required power that must be supplied by the Bosch heaters, but the reaction never generates enough heat to be self-sustaining.

CO_2 REDUCTION DESIGN

1. **Bosch maximum capacity for 1 unit-** 4 astronauts or 8.8 lb of CO_2 per day.

2. **Modes of operation of the CO_2 Assembly**

a. Normal mode- The Bosch is reducing CO_2 to water & Carbon(S). The single pass efficiency is less than 10%, so the reactants have to be recycled and passed back through the Bosch. Less than one percent of the exiting reactants are not reduced to water & solid Carbon. The Bosch will operate for 90 days before servicing the Carbon Filtration Cartridge is necessary.

b. Standby mode- Everything is powered & ready to go except all valves are closed and the compressor is off.

c. Shutdown mode- The heater & compressor are off and all valves are closed. All sensors are working.

d. Purge mode- The system is being purged with nitrogen. The purge is drawn off through the nitrogen purge/bleed vent. The compressor & heater are off.

e. Unpowered mode- No electrical power is applied to system.

3. **Process startup-** The process starts in the unpowered mode and switches to the shutdown mode. The system is checked for leaks and if none are detected the Carbon Dioxide Reduction Assembly is purged with Nitrogen. While the Assembly is being purged, the heaters in the Bosch are turned on and kept at a constant 200 °F for two hours. This is to drive off any moisture accumulated in the Bosch during servicing of the Carbon Filtration Cartridge. After the two hours the leak check and purge are finished and the heater temperature is increased to 1050 °F. The compressor is started and the purge Nitrogen is circulated around the system. When the reactor temperature reaches 1050 °F the reactants are introduced. The average time from leak check to normal operating mode is 12 hrs.

	Design Point	Range
Metabolic CO_2		
Flow Rate,lb/day	8.8	8.80-17.60
Temperature,°F	70	60-85
Pressure,psia	18	14.7-20
H_2 Feed		
Flow Rate,lb/day	.80	.80-1.60
Temperature,°F	75	75-100
Pressure,psia	30	14.7-30
Product Water		
Flow Rate,lb/day	7.20	7.20-14.40
Temperature,°F	60	60-90
Pressure,psia	30	14.7-30
Bleed		
Flow Rate,lb/day	1.12	1.12
Temperature,°F	75	65-90
Pressure,psia	18	14.7-20
Electric Power		
28 VDC,W	341	306-606
115 AC,W	186	170-3120
Heat Rejection,W		
To Air	529	494-818
To Coolant	238	181-461

Table 2.2-1 Flow Rates of Inputs[2]

[1] "Carbon Dioxide Reduction Description", Boeing Aerospace & Electronics. Huntsville, Alabama. April 20, 1990, Doc # 2-H8RG-RJK-198.

[2] "Bosch Carbon Dioxide Reduction (Bosch - III) Subsystem", Volume 1, Life Systems Inc. October 1987.

2.3 Temperature and Humidity Control Subsystem

INTRODUCTION

Because the CO_2 Removal Assembly model takes air from the Temperature and Humidity Control (THC) Subsystem, it was necessary to research the THC Subsystem and determine the effects it may have on the air entering the CO_2 Removal Assembly.

DESCRIPTION

The principal function of the Temperature and Humidity Control (THC) Subsystem is to maintain a comfortable environment for the astronauts in Space Station Freedom (SSF). Air in the cabin is continuously circulated through the THC system at 340 cubic feet per minute. Temperature and Humidity are controlled by a condensing heat exchanger and a slurper, respectively. Depending on the temperature change needed to keep the cabin air within certain specifications, a temperature control valve determines the amount of air passed through the condensing heat exchanger. Air not passed through the heat exchanger is bypassed to the exit side of the heat exchanger. As air passes through the heat exchanger, water vapor is condensed and drawn into a slurper to remove excess water vapor in the air and prevent excessively high cabin humidity. After leaving the heat exchanger, the air is pulled across a mixed flow fan, and passed through an air straightener before it is returned to the cabin.

EFFECTS ON CO_2 REMOVAL ASSEMBLY

According to available THC documentation, air drawn by the CO_2 Removal Assembly is taken from the THC system immediately after the temperature control valve, and returned just before it reaches the air straightener. This configuration is not acceptable for the following reasons. First of all, the air drawn by the CO_2 removal assembly cannot be taken after the temperature control valve because in some cases, the control valve may bypass all air flow around the heat exchanger leaving none available to the CO_2 Removal Assembly. Secondly, because the air returned from the CO_2 Removal Assembly may not be within the established cabin parameters, it should not be returned after the THC air conditioning process.

The model will take air from the high pressure side of the THC fan and return it to the THC Assembly at the inlet. Two assumptions will correspond to this configuration. Because air is taken from

the THC after the heat exchanger, inlet conditions into the CO_2 Removal Assembly will be assumed to remain within cabin parameters. Secondly, the flow rate of air from the CO_2 Removal Assembly will be assumed much smaller than that being drawn from the cabin so the retruned air cannot cause a false response in the temperature control valve.

CABIN PARAMETERS

The cabin pressure operates between 14.5 and 14.9 psia, and the temperature will be kept between 65 and 80 °F. Relative humidity and partial pressure of CO_2 will be maintain within 25 to 75 % humidity and 3 to 12 mm of mercury, respectively.

The only cabin parameter which will act independently of the THC system is the production of CO_2. A schedule which approximates the production of CO_2 by the astronauts for a 24 hour period is described in Section 2.4.

The effects of these parameters on the model will be tested within and outside of the ranges given.

2.4 Cabin Model

DESCRIPTION

A model of the cabin was produced to simulate the effects of CO_2 production and removal on the cabin atmosphere. The model simulates temperature, pressure, and relative humidity levels within the cabin by three different functions: a constant value, sinusoidal and step functions varying within specified parameters. For each time step, the model evaluates the amount of CO_2 produced within and removed from the cabin and determines the current partial pressure of CO_2 inside the cabin.

MATH MODEL

The model allows for the cabin conditions of temperature, pressure, and relative humidity to be simulated in several ways. Relative humidity and temperature can be varied by use of either a sine or step function and will fluctuate between any given parameters establish within the program. The model also allows for varying the cycle time of each function. Because normal cabin pressure conditions only fluctuate between 99.9 and 102.7 kPa, the pressure is only simulated by either a constant value or a sine function. After the values for the cabin pressure, temperature, and relative

humidity are determined for a particular time step, the program evaluates the amount of CO_2 produced by the astronauts for that time step.

Figure 2.4-1 shows the CO_2 production within the cabin per 24 hour period. Beginning at midnight, time zero, the CO_2 production levels are at the lowest value because the astronauts are sleeping. At 0600 hours, all four crew members awake and for the next four hours each takes an hour of exercise, producing the highest level of CO_2 procudtion. From 1000 to 1400 hours, normal breathing processes are maintained as the crew performs the daily duties, and from 1400 to 1800 hours each memeber again takes an hour of exercise. At 1800 hours normal activities are resumed until the astronauts go to bed at 2200.

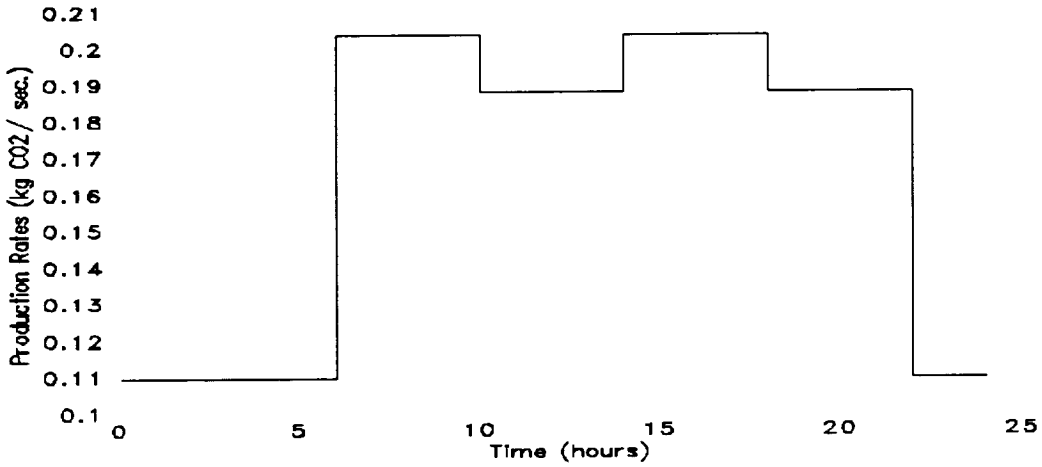

Figure 2.4-1: Cabin CO_2 Production

Next, the model calculates the mass of CO_2 in the cabin by taking the previous amount of CO_2 in the cabin, adds the mass produced and subtracts the amount removed for that time step. The mass of the air at the current cabin conditions is determined using the ideal gas law in Equation 1 and stated as

$$M_{air} = \frac{(CPress)\,(CVol)}{(R)\,(CTemp)}. \qquad \textbf{(1)}$$

The symbols used in the above equations are defined as follows:

M_{air} = mass of air in the cabin,
CPress = current cabin pressure,

CVol = cabin volume (101.115 m^3),
R = gas constant (8.314/29),
CTemp = current cabin temperature (°C).

The mole numbers for the cabin air and the CO_2 in the cabin are calcuated by dividing the mass of the air and CO_2 by the molecular mass of air and CO_2, respectively. Next, the mole fraction of CO_2 to air is calculated by dividing the number of CO_2 moles by the number of moles of air, and the paritial pressure is calculated by multiplying the current cabin pressure by the mole fraction. The caculated cabin partial pressure of CO_2 is then checked to determine if the CO_2 removal assembly needs to be turned on to remove any excess CO_2.

3.0 CONTROLS

3.1 Classical Control

DESCRIPTION

The CO_2 removal sub-assembly is responsible for maintaining the partial pressure of CO_2 with in normal limits as the astronauts and other equipment and experiments produce it. NASA grades air quality by the partial pressure of CO_2, with normal CO_2 pressure being 0.0667 kPa. When the CO_2 partial pressure is above 0.4 kPa the air is classified as "degraded" and above 1.015 kPa the condition is classified as "emergency". The CO_2 removal sub-assembly removes CO_2 from the cabin environment and stores it as a gas in a CO_2 accumulator tank until the Bosch reactor breaks it down to solid carbon and water.

The CO_2 removal sub-assembly uses a variable speed fan to force air through the system's beds, ducts and heat-exchangers. The desiccant beds and the CO_2 sorbent beds operate on 30 minute cycles, where one bed adsorbs mass for 30 minutes while the companion bed is desorbing. After 30 minutes the beds reverse roles and the full adsorbing bed desorbs its mass while the empty desorbing bed adsorbs mass.

CLASSICAL CONTROLS

There are two inputs that control the operation of the CO_2 removal sub-assembly, the partial pressure of CO_2 in the cabin and the pressure of CO_2 in the CO_2 accumulator tank. The cabin CO_2 pressure input is used as input to a classical control to maintain the cabin CO_2 pressure. If the partial pressure of CO_2 in the cabin deviates from the desired 0.0667 kPa the system would modify the air flow rate.

The input from the CO_2 accumulator tank was based on the gas pressure in the tank. The Bosch reactor is an important producer of fresh water and a shortage of CO_2 may mean a corresponding shortage of fresh water. The Bosch reactor shuts down if the pressure of the supply CO_2 (the CO_2 tank) dips below 101.125 kPa, so the system is turned on if the pressure in the CO_2 accumulator tank drops below 137 kPa. This safety buffer of 36 kPa assures that the tank pressure should not go below the lower limit of 101.125 kPa.

Internal to the CO_2 removal sub-assembly are controls that maintain

the pressure of the CO_2 accumulator tank and a valve that is positioned before the CO_2 accumulator tank and after the CO_2 pump that controls the purity of the CO_2 entering the tank.

The cabin air is driven through the system by a variable speed, zero-inertia fan that is controlled to maintain cabin pressure of 0.0667 kPa. Classical control of the fan speed is accomplished by using a proportional-integral-differential (PID) compensator in a negative feedback loop. The PID compensator uses an error function δ, defined as the difference between the actual CO_2 cabin pressure and the desired cabin pressure. The magnitude of the change in the pump speed is given as

$$\Delta_{fanspeed} = \delta + \frac{d\delta}{dt} + \int \delta \, dt. \qquad (1)$$

The fan speed is then adjusted by this amount, increasing or decreasing the tank pressure.

The valve between the CO_2 accumulator tank and the CO_2 pump serves two purposes. One is to direct CO_2 gas to the accumulator tank when the pressure in the desorbing CO_2 sorbent bed is within 1% of the equilibrium pressure of CO_2 for the bed. This insures that the gas that is directed to the CO_2 tank is almost entirely CO_2. The other purpose is to quickly evacuate the air from the CO_2 sorbent bed that just switched to the desorbing cycle. As the beds switch, the full bed that is just beginning to desorb contains cabin air and CO_2 trapped in the absorbent material. For the first several minutes of the desorbing cycle the gas removed from the bed is air and, as the pressure in the bed decreases as the air is removed, the temperature of the bed increases and the equilibrium pressure of the CO_2 trapped in the Zeolite begins to increase. As the pressure of the bed and the CO_2 equilibrium pressure converge, the purity of the CO_2 gas leaving the bed increases. When there is a difference of greater than 1% between the pressures, the valve directs the gas back to the exit gas from the adsorbing CO_2 sorbent bed and turns the CO_2 pump to its maximum speed to expedite the emptying of the bed.

3.2 Expert Systems Control

IMPLEMENTATION

The simulation of the Carbon Dioxide Removal Assembly can be controlled by an expert system written in CLIPS using fuzzy logic. The simulation for the physical system is written in FORTRAN. The purpose of using FORTRAN is that an existing FORTRAN simulation has already been developed by mechanical engineers of the NASA group. Last semester we struggled with choosing between C and FORTRAN as a simulation language. The simulation equations were taken from the existing FORTRAN simulation and implemented in C. The C model then communicated with CLIPS to make a separate model apart from the classically controlled simulation.

Originally the computer science students felt that it would be easier to integrate CLIPS into the C environment. Since they would be implementing the CLIPS program into the simulation, they felt they should work with a program that was most familiar to them, hence C. Later on when the expert control was running, it was found that it was a painfully slow working with the simulation. This is caused by too much file I/O overhead. The reason for this is that CLIPS cannot communicate or link to any programming language other than itself. The problem is sharing variables between two languages. On the one hand, we did not want to implement the whole model within CLIPS. It is not that easy to program a simulation using an expert systems programming language. Rules do not get fired in the order that one expects. On the other hand, we did not want to implement the whole system in C either. Programming a recursive expert systems controller in C can be quite a struggle. It would be easier to use an expert systems program that was designed to do just that. Therefore, we were left with the job of integrating the two into one environment. Initially we used a file sequencer that monitored the reading and writing of the variable files between C and CLIPS. This was extremely slow and used about 90% of the processor of a Soulbourne SPARC computer running UNIX System Release V.

Since the original design was slow and the system administrators of the computer resources weren't happy that it took so much processor time, we ventured to design a new system. Semaphores was one of the solutions that were brought up, but no success on implementation was ever achieved. A semaphore is a process that gets "forked" off from the initial process. What a semaphore does is protect what is known as a critical section. In our case, the critical section is the file being passed between CLIPS and C. This file contains all of the variables pertinent to the running of the simulation. Some of the variables simply get read by the processes, and other get changed in the process. However, most, if not all, variables in this file get changed at one time or another.

The problem is that not both processes can be writing to this file at the same time. This would cause chaos. So this is where a semaphore becomes useful. A semaphore would allow the C program to start its simulation process, and when it was ready, "fork" off the expert control CLIPS process. You then tell the semaphore where the critical section of code exists (the reading and writing of the variables file), and then both processes can run simultaneously waiting for the next one to hand control to it. Again, a semaphore in this case would keep the variable file from being written to at the same time by two different processes. However, as stated before, a working implementation of semaphores was never worked out.

So then we move onto the third and final design which we are currently using. The C simulation was dropped and the FORTRAN simulation was used for both the expert control and the PID control to maintain a completely consistent environment.

At this point we are in the Spring 1992 semester of implementation of the project. We have decided to use FORTRAN as the simulation, C and Xwindows for a user interface, CLIPS for the expert systems controller, and FORTRAN for the PID controller. We have also decided to use a Soulbourne SPARC UNIX workstation as our platform of choice. Our reasons behind this are simple; it is an extremely fast computer, it multitasks, and hard disk space on this system is plentiful.

The engineers programmed the FORTRAN simulation and PID controls while three computer scientists programmed the expert systems control and the Xwindows user interface. Much of the semester was spent by the engineers getting a working "bug free" simulation running so the controllers could be employed. The expert systems controller was built within the first month and then modules were stubbed for tests. After this, efforts were placed on getting the user interface to work.

There is not much to the expert systems in terms of lines of code. However, this does not mean that it does not do a lot. Some of the expert system was designed like the PID controller because there was really no expertise that could be used in making a decision. Situations such as a valve require only two positions, i.e. on or off. Other such devices require only If/Then statements, and no fuzzy logic was used in determining what variables to change. For the devices that can take advantage of fuzzy logic, CLIPS becomes very powerful. Once the engine to determine membership functions has been written, it can be used over and over to control many different devices. All that need be added to it are one or two lines at the beginning of the CLIPS code that describe what ranges the variables should exist in. The engine takes care of the rest. One such example might be as follows:

```
(fuzzy tank-pressure    low -150 300 400),
```

```
(fuzzy tank-pressure     high 300 400 650),
(fuzzy cabin-pressure    low  -100 .058 .075),
(fuzzy cabin-pressure    high .058 .075  100).
```

Presently, the only component within the CO_2 Removal Assembly which is controllable is a variable speed fan used to draw mass through the sorbent beds, which remove carbon dioxide from the atmosphere. The controller monitors the Cabin CO_2 partial pressure to determine when the pump speed should be adjusted to maintain safe CO_2 levels.

PRESSURE CONTROL BY EXPERT SYSTEM METHODS

The expert system uses triangular functions to control the simulation. A triangular function consists of three values: low, medium and high, as shown in Figure 3.2-1.

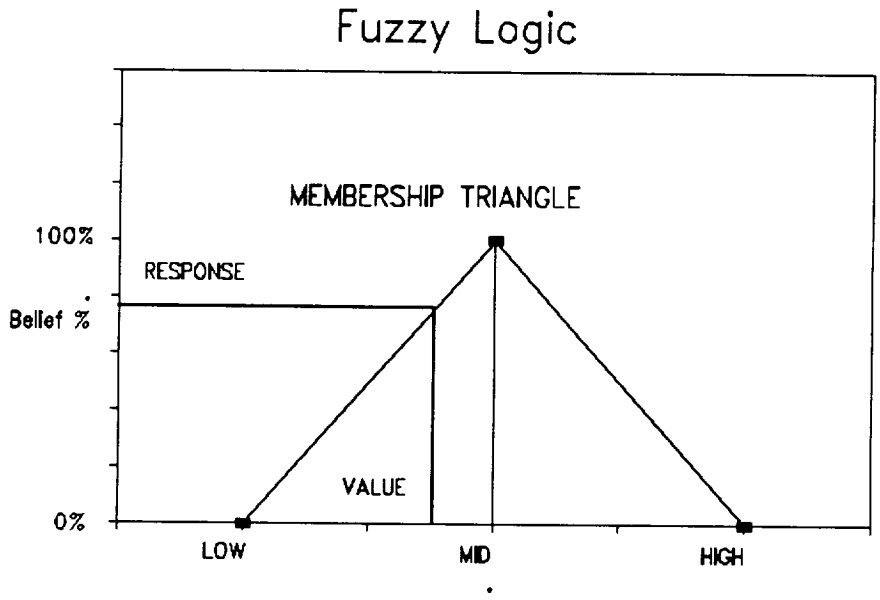

Figure 3.2-1 Fuzzy Logic Membership Triangle

A function is used to calculate a percentage belief when the value being considered is in the range low to high. When the value does not lie in the range low to high, the percentage belief is zero. A belief is calculated with Equation 1 when the value being considered is in the range low to mid, and Equation 2 is used when the value is in the range mid to high. Given by

$$\frac{value-low}{mid-low}, \tag{1}$$

and

$$\frac{high-value}{high-mid}. \tag{2}$$

The percentage belief is used to directly determine the amount of change that must be made. This expert system uses two triangles to control the simulation. The left triangle represents the low pressure function. The right triangle represents the high pressure function. There is also overlap between the high and low triangles. This is not uncommon in fuzzy logic. The intersection point of the two triangles is chosen to correspond to the target control value and to a 50% belief in both triangles. This is done so that when the system variable deviates from the target value, the belief is immediately greater than 50% in one of the triangles which prompts the system to try to correct it. The slope of both triangles is adjusted to control the rate at which the expert system changes the simulation. Pump speed, pump duration, and pressure deviation are factors used in determining the adjustments to the triangular functions. The pressure can be controlled more accurately when the pump speed is changed more often. However, this can cause wear on a pump and must be taken into consideration. The definition of the functions in CLIPS are as follows.

```
(deffacts start
     (state open)
     (fuzzy temp low    0 258 338)
     (fuzzy temp high 268 348 600))
```

When the percentage belief in a low pressure is greater than 50%, Equation 3 is used to adjust the pump speed. Likewise, when the percentage belief of a high pressure is greater than 50%, Equation 4 is used to change the pump speed. In this way the expert system is able to control pump speed by monitoring the tank pressure as given by

$$NewPumpSpeed=OldPumpSpeed\times(1+\%beliefcold), \tag{3}$$

$$NewPumpSpeed=OldPumpSpeed\times(1-\%beliefhot). \tag{4}$$

After trial runs were executed using these equations, it was decided to adopt a more fluid control equation. It employs a normalized belief, and is less prone to overshoot and repeated searching for the desired value. Equation 5 shows the method of

employing this normalized belief as .

$$NewPumpSpeed = OldPumpSpeed \times (1 + (2 \times \%beliefcold - 1)). \quad (5)$$

This improved control equation was then adopted into all full simulation exercises.

The expert system and simulation communicate by writing a temporary file. The two programs work in lock step. In other words, one program runs one cycle, then the other program runs one cycle. The FORTRAN simulation was modified to run only one time step and then shell out to the operating system to call CLIPS. The simulation will have to read in variables from a file each time that it runs. It must then save the variables to the same file after each run. The expert system will be acting in the same way with one exception. Instead of running continuously, the expert system will run only once, make the necessary changes to the file variables, and then exit; thus handing control back over the simulation program.

4.0 DYNAMIC SYSTEM SIMULATION

4.1 Introduction

DESCRIPTION OF THE CO_2 REMOVAL SUB-ASSEMBLY

The objective of the CO_2 Removal Sub-Assembly is to remove carbon dioxide from cabin air and store it in an accumulator tank for further processing. The sub-assembly consists of nine main components: two desiccant beds, a blower, a pre-cooler, two sorbent beds, a CO_2 pump, a CO_2 accumulator tank, and control schemes. The function of each sub-assembly component is described below.

The desiccant beds dehumidify used cabin air, and humidify fresh cabin air. This is done simultaneously by two desiccant beds working together at a set operating cycle. While one bed is removing water from used cabin air, the other bed is releasing water to carbon dioxide free, fresh cabin air.

The blower moves the cabin air through the desiccant beds and on to the sorbent beds. The pre-cooler cools the dry cabin air from the desiccant beds to a uniform temperature.

The sorbent beds adsorb carbon dioxide and desorb stored carbon dioxide. This is done simultaneously by two sorbent beds working together at a set operating cycle. While one bed is removing carbon dioxide from dry cabin air, the other bed is releasing stored carbon dioxide to the CO_2 accumulator tank.

The CO_2 pump draws off the stored carbon dioxide from one sorbent bed and passes it on to the accumulator tank. The CO_2 accumulator tank stores released carbon dioxide from the sorbent beds, and passes it on to the CO_2 Reduction Sub-Assembly.

The control schemes regulate the speed of the blower based on the information on the CO_2 level in the cabin. Two different control methods are used. The first discussed is a classical method using a PID approach. The second method uses an expert system and fuzzy logic to accomplish control over the blower.

DEVELOPMENT OF THE SIMULATION PROGRAM

The objective of the simulation program is to accurately model and control the main components of the CO_2 Reduction Sub-Assembly over a set operating time using a Fortran code program. The simulation program is composed of a main simulation program and

several subroutines that model the main sub-assembly components.

Before a main program was written, individual subroutines were written to model each main sub-assembly component. After the subroutines worked successfully on their own, they were integrated together using the main program.

The function of the main program is to initialize all variables that are common to two or more subroutines, and pass these variables through CALL statements. By initializing most of the variables in the main program, any changes to these variables can be done by accessing the main program only, and not each individual subroutine. Any variables that are common to only one subroutine were kept localized within that subroutine. The main program operates on a set operating cycle at a constant time step. Currently, the operating cycle of the main program is set at 24 hours or 84600 seconds, and the time step is set at one-tenth of a minute or 6 seconds. The name of the main program is COOL.FOR

The function of the subroutines is to calculate common subroutine variable values for a given time step. By keeping all computations within the subroutines, it is easier to locate, assess and adjust erroneous data. The names of the subroutines are as follows:

desiccant beds: DESSBED.FOR
blower/pre-cooler: BLOWCOOL.FOR
sorbent beds: SORBED.FOR
sorbent beds/CO_2 pump: SORPUMP.FOR
accumulator tank: CO2TANK.FOR

Analysis of the entire CO_2 Removal Sub-Assembly reveals 4 state variables for each main component. Given as

1) mass flow rate,
2) pressure,
3) temperature,
4) relative humidity.

These state variables are coded and localized for each subroutine by the type of variable it is (mass, pressure, temperature, relative humidity), which subroutine it is in (DESSBED, BLOWCOOL, SORBED, SORPUMP, CO2TANK), and whether it is at the inlet or outlet (in, out). For example in the DESSBED subroutine, mass flow in is represented by M for mass, DB for desiccant bed, and IN for inlet, yielding MDBIN. Similarly, inlet pressure and temperature are PDBIN and TDBIN and outlet mass, pressure, and temperature are MDBOUT, PDBOUT, and TDBOUT. Following this pattern, the following variables were coded:

blower/precooler: MBPIN, PBPIN, TBPIN, MBPOUT, PBPOUT, TBPOUT
sorbent beds: MSBIN, PSBIN, TSBIN, MSBOUT, PSBOUT, TSBOUT
pump/tank: MPTIN, PPTIN, TPTIN, MPTOUT, PPTOUT, TPTOUT

When passing variables from one subroutine to the next, the outlet variable for one subroutine will become the inlet variable for the following subroutine. As a result, mass out from the desiccant beds becomes the mass in to the blower/precooler. Consequently, the main program passes the previous outlet variables in the CALL statements, but receives them with the inlet variables in the subroutines.

4.2 Classical Control Results

INTRODUCTION

The simulation with controls needed to be thoroughly tested. This would result in two benefits. First it would be possible to determine if the physics of the CO_2 removal process were being correctly modelled. Second, it would allow an insight into the abilities of both the system and the controllers to handle various situations.

The method used of evaluating the control systems was to determine which "weighting factor" provided the most desired response. The major characteristic looked for in the solution was the ability of the controller to dampen out initial transients, and settle upon a closely bound mass flow rate and therefore CO_2 rate. This resulted in the system being run at a nearly constant rate which greatly reduces wear on the fan due to cycling.

Although many tests were run, the test conditions used for the evaluation of the controllers was a simple twin step function with an initial offset. It was desired to maintain cabin CO_2 at 0.0667 kPa throughout the test. The initial value in the cabin was set at 0.07 kPa. The CO_2 production rate was initially given as $1.7*10^{-5}$ kg/sec, indicative of resting astronauts. At four hours into the simulation this value was increased to $7.0*10^{-5}$ kg/sec, a number representing a double sized crew performing hard work. Finally at eight hours the level was decreased to $3.0*10^{-5}$ kg/sec a level appropriate for the standard 4 man crew performing typical functions.

DESCRIPTION

The classic, or PID, controller was designed around the corrective algorithm given by

$$\dot{m}=\dot{m}+\left(\frac{error}{k1}+\frac{\frac{d}{dt}error}{k2}+\frac{\int errordt}{k3}\right), \quad (6)$$

where m refers to the mass flow rate through the blower. In its initial form the values of k1, k2, and k3 were all equal to unity. This resulted in two major effects. First the controller was able to quickly vary the flow rate resulting in the controller exhibiting a very high frequency. Second the derivative's terms influence was very small. The Figure 4.2-1 shows this controller's response to the test conditions detailed in the preceding paragraph. The partial pressure of CO_2 in the cabin corresponds to the top curve and is scaled along the right hand axis. The mass flow rate through the system is the bottom curve, and is scaled along the left hand axis.

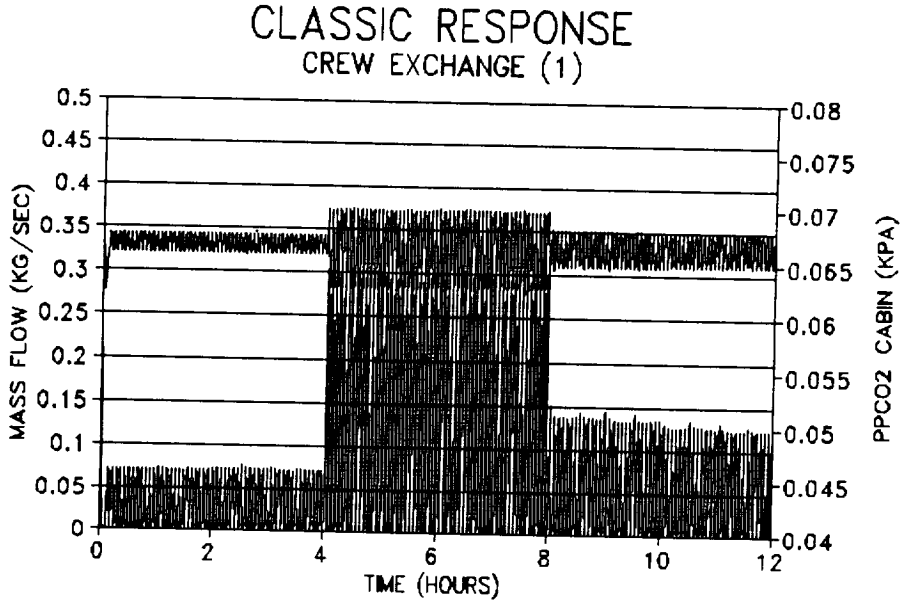

Figure 4.2-1 System Response with Weighting (1,1,1)

This figure obviously has little if any dampening evident, and so this initial set of constants scored poorly on the scale of desirability. This led to the need to increase the impact of the derivative term, and also to lower the frequency of the controller as the original constants lead to value searching at

unrealistic rates.

For a second try the value of k1 and k3 were increased to 10. This would result in a slower frequency due to the controller changing the mass flow at a slower rate, and a better dampened system as the relative impact of the derivative term would be increased. The results of this controller when subjected to a similar test are shown in Figure 4.2-2. This controller was able to achieve an appreciable amount of dampening during the four to eight hour interval corresponding to the highest CO_2 production rate. However, at other times it was unable to achieve dampening and so this set of weighing factors did not represent a satisfactory solution.

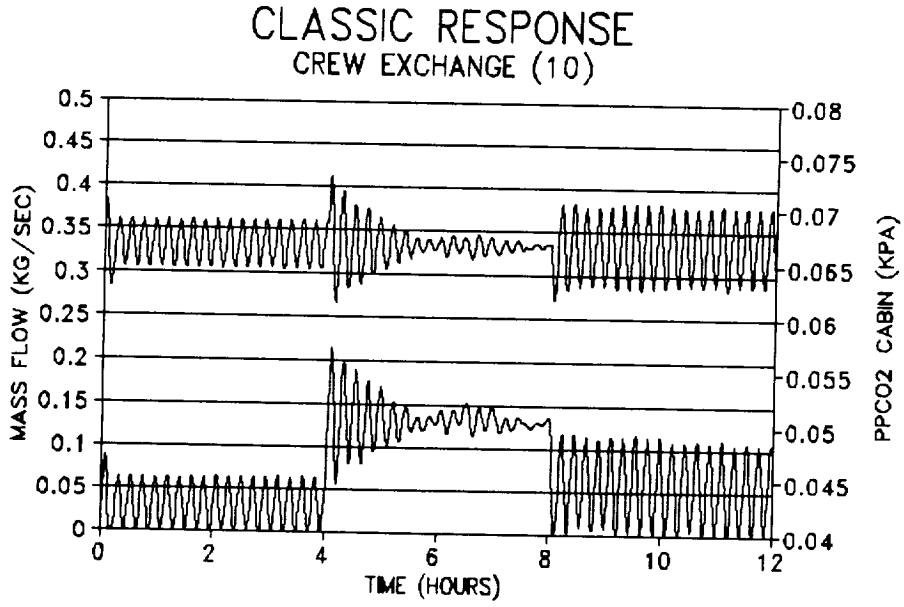

Figure 4.2-2 System Response with Weighting (.1,1,.1)

The next attempt was with the value of k2 still at 1 and the values of k1 and k3 set at 30. The system response is shown in Figure 4.2-3. Here we see some significant dampening, and the mass flow rate stays relatively well bounded. This controller could be labelled as acceptable, but it was decided to see if a further improvement could be found.

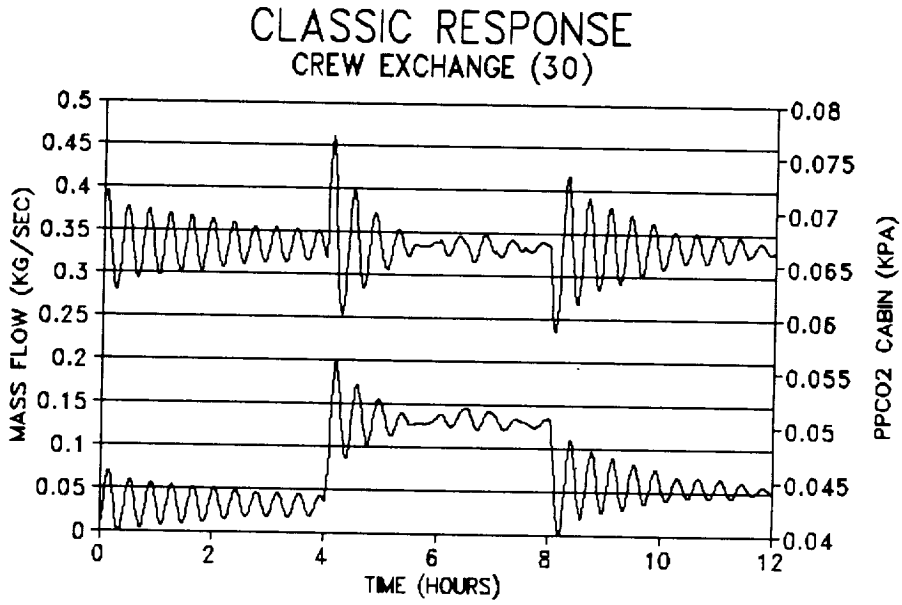

Figure 4.2-3 System Response with Weighting (.03,1,.03)

The trend would indicate that increasing k1 and k3 results in a better controller. In pursuit of that trend the next controller was run with the weighing factors even larger. The constant k2 was again left at 1 to provide dampening, while k1 and k3 were increased to 50 to reduce the controller's frequency. The results from this test were largely similar to the results from the controller run previously with a few minor differences, and are given in Figure 4.2-4. First the transient spikes in partial pressure from the step changes were a little larger, though still easily acceptable. Second this controller though not as capable at dampening during the big CO_2 production period, it was a more effective controller during the final four hour period.

There is no reason that the value of k1 and k3 had to be left equal to each other. Since the system was well behaved and smooth, it was not necessary to incorporate a large integral term. This fact allows us to assign a very large value to k3 and in essence reduce the PID controller to a nearly PD controller. By reducing the input from the integral term, it was possible to increase the contribution of one of the remaining terms and maintain a similar controller.

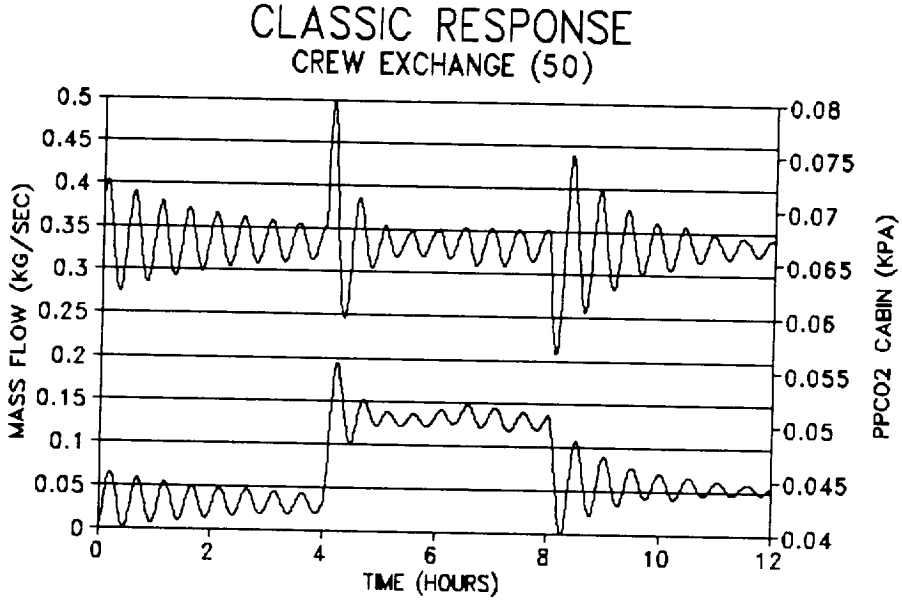

Figure 4.2-4 System Response with Weighting (.02,1,.02)

Since the value of k2 was already fairly small, it was decided to decrease the value of k1 back to 25 to increase the effectiveness of the proportional term. The net result was a controller with the constants set at k1 = 25, k2 = 1, k3 = 100. These constants do not represent a calculated attempt at optimizing the controller, rather a logical qualitative approach to examine the effect of the different error terms on the overall responses to the test. The data for its response to the test case is shown in Figure 4.2-5.

This controller exhibits several characteristics. First it suffers from a large spike in partial pressure corresponding to the onset of the step functions. The maximum value attained was 0.084 kPa of CO_2. The duration of the spike was for only a few minutes, and is not a problem to the crew. On the positive side, this controller was able to quickly reduce the magnitude of the oscillations and rapidly achieved a steady mass flow rate. Comparing these results to our previously listed criteria, this set of constants was elected as best for use in the classic PID controller.

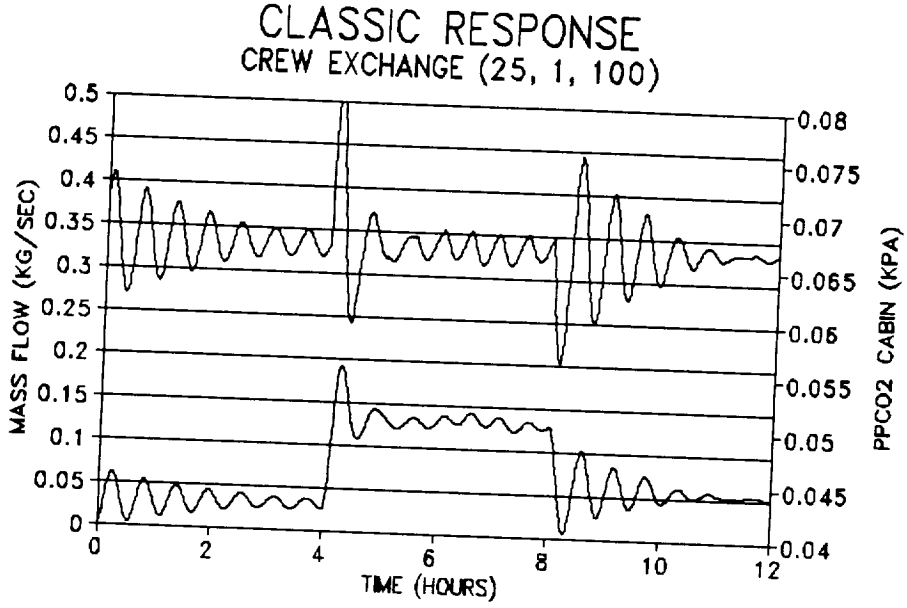

Figure 4.2-5 System Response with Weighting (.04,1,.01)

The loading in the sorbent beds during this exercise was also of interest. It was desired to confirm that the system was not experiencing a problem with residual loading in the beds. The sorbent beds require that the disturbing cycle removes enough CO_2 that they don't simply continue to load until they reach a saturation. Figure 4.2-6shows the bed loading curves with the percentage loading of both beds plotted versus time. From the graph it is easy to see that the beds are not suffering any residual loading problems.

The PID controller was very successful in regulating the system and maintaining desirable cabin conditions. The effect of the constants on the response of the system as expected lending an air of credibility to the model and the controller. Again let it be understood that the controllers tested were chosen in search of a capable and satisfactory controller, not the result of a formal optimization study.

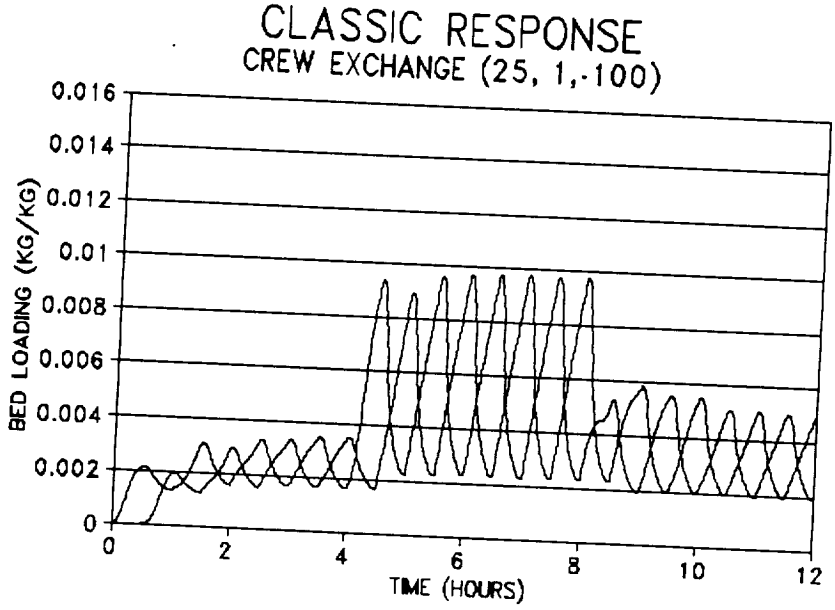

Figure 4.2-6 Bed Loading Curves

4.3 Expert Control Results

DESCRIPTION

The expert controller was subjected to testing using the exact same cabin conditions as described in Section 4.2 called Classical Control Results. It was neccessary to again attempt to modify the expert controller to provide some degree of dampening to lessen the wear on the fan and motor driving the air through the sorbent beds. The understood restraint on maximizing dampening is that the system must maintain the cabin CO_2 levels at approximately the 0.0667 kPa set point.

The expert system algorithm first generates a belief, a percentage basis of its need to execute a change. This belief is multiplied by a weighting factor to generate a new mass flow rate. The actual algorithm is presented as
The most obvious characteristic of this equation is that the controller's frequency is proportional to K1 or the weighting factor. That is, a large factor will generate a high frequency

$$\dot{m}=\dot{m} \pm (k_1(2 \cdot \%Belief-1)). \quad (7)$$

controller. The inverse of this is that a small weighting factor will result in a lower frequency controller.

The original controller was designed with k1 equal to 0.05. The result of this controller when tested with the crew exchange scenario is given as Figure 4.3-1. The upper curve corresponds to the right hand axis and displays the partial pressure of CO_2 in the cabin in kPa. The left hand axis goes with the lower curve to show the mass flow rate in kg/sec.

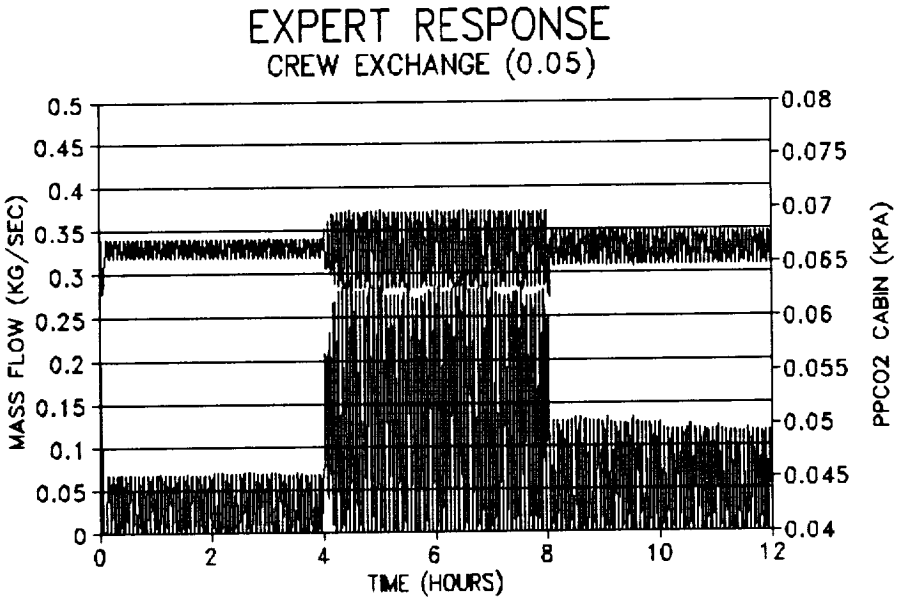

Figure 4.3-1 Dynamic Response with Weighting (0.05)

The controller exhibits no apparent dampening, and so does not appear very suitable for our application. The next course of action was to remember, as with our work on the PID controller that a lower frequency controller provided smoother mass flow rates and an increase in dampening. Following that hunch, the value of K1 was lowered to 0.02 and the test was run again.

The results for the test at k1 = 0.02 are given in Figure 4.3-2. There is still no evidence of dampening, and the only major deviation between the two runs was the fact that the second controller was not able to keep the partial pressure of CO_2 as

tightly regulated as the first quicker controller. Therefore both of these controllers were declared unsatisfactory.

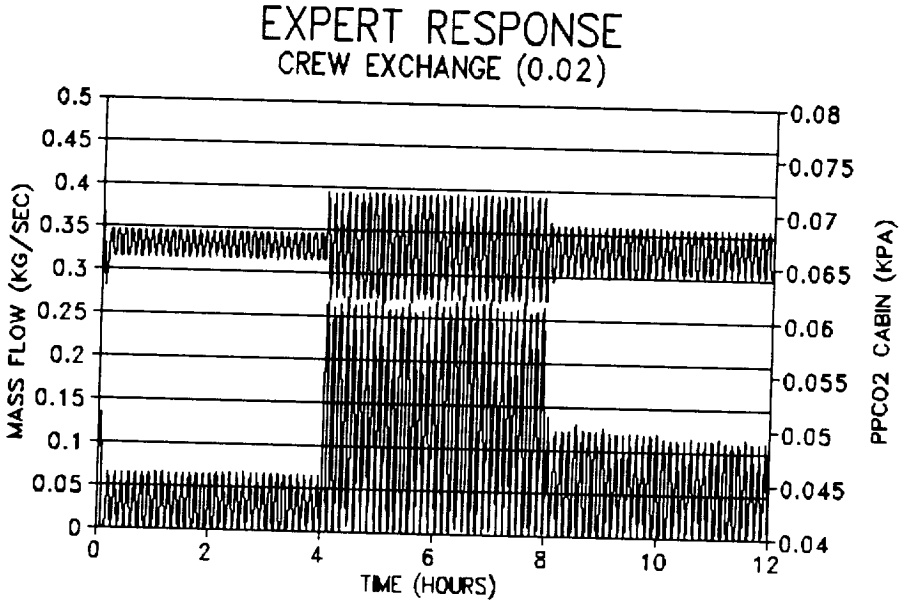

Figure 4.3-2 Dynamic Response with Weighting (0.02)

The next trial was conducted with an even smaller value assigned to k1. For this test the weighting factor was reduced all the way to 0.005. This served to slow the controller's time of response, and also to achieve a slight dampening effect. The results for this run are shown in Figure 4.3-3. The quickest dampening however was limited to the region when CO_2 was the highest. This trend was similairly observed in the PID controller when the frequency was slightly too high. This indicates that the weighting factor is close to the desired value, and only needs fine tuning.

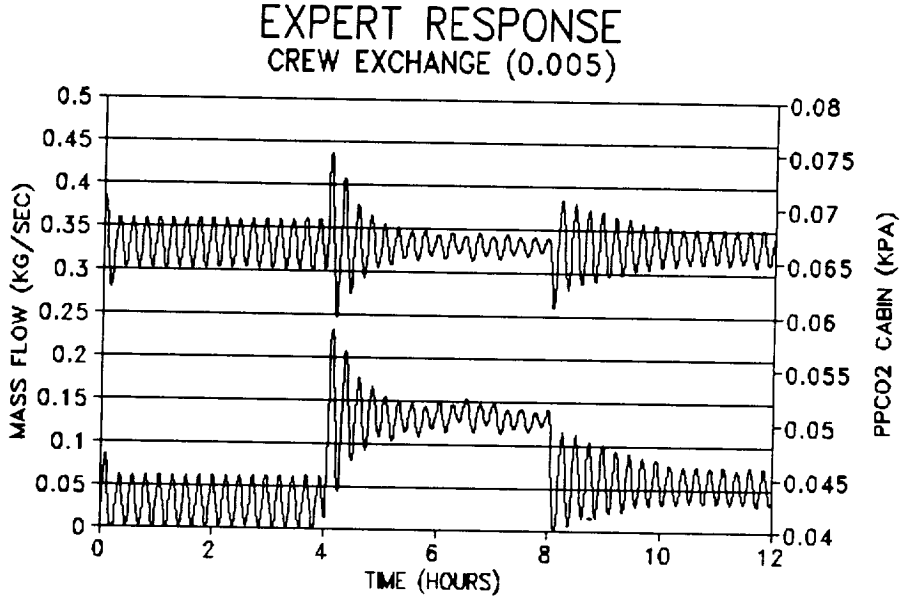

Figure 4.3-3 Dynamic Response with Weighting (0.005)

The next K1 was then given a value of 0.001 and this test data was subjected to the same situation. The graph portraying these test results can in Figure 4.3-4. Here the dampening that was desired becomes apparent at every level of CO_2 generation. This controller however has one major handicap. It was too sluggish to appropriately react to the transient cases. The peak at four hours and the dip at eight hours both represent undesirable deviations from the 0.0667 kPa set point. These deviations are short lived, and do not represent a problem for the human occupants. The net result being that this is an acceptable controller as it meets the basic criteria.

The final variation on the expert system weighting factor was to set k2 = to 0.002. The graph in Figure 4.3-5 represents the results of that test. It can be seen that the increase in controller frequency enabled the controller to decrease the amplitude of the transient spikes. That reduction coupled with the fact that the dampening was even more successful made the weighting factor of 0.002 appear to be the most capable option for the expert controller. The sorbent bed loading curves for the test run at k1 = 0.002 are also included in Figure 4.3-6.

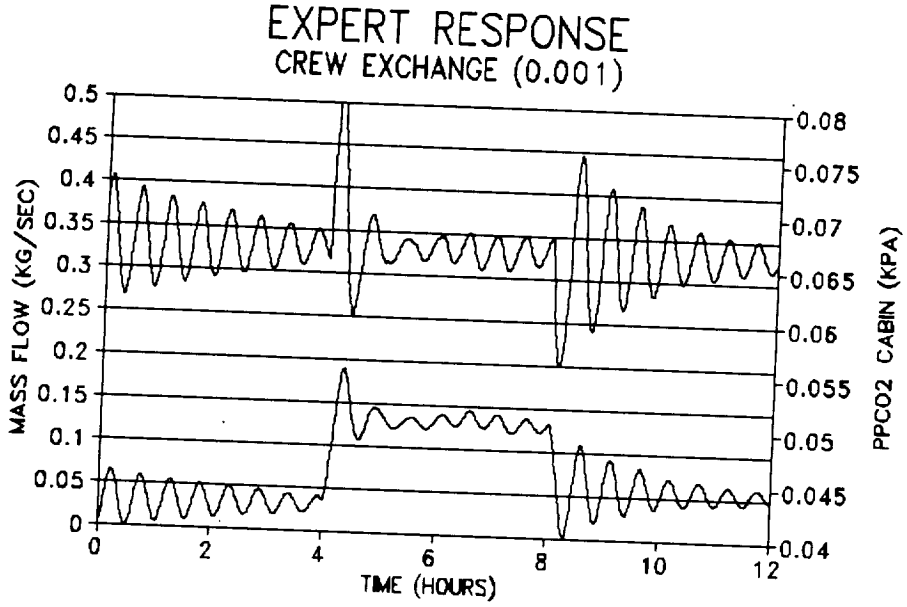

Figure 4.3-4 Dynamic Response with Weighting (0.001)

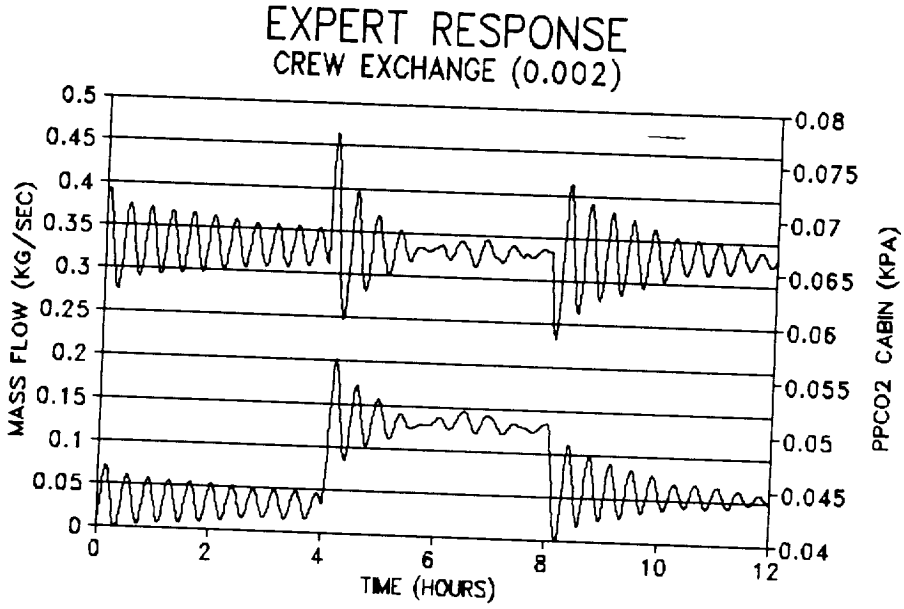

Figure 4.3-5 Dynamic Response with Weighting (0.002)

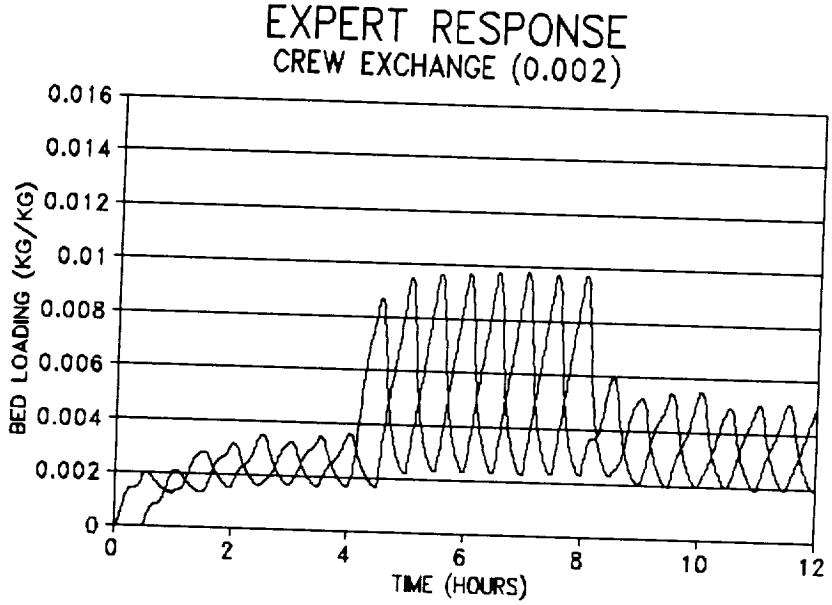

Figure 4.3-6 Bed Loading

Again it is important to stress that the expert controller like the classic controller is not optimized. Although the apparent best choice from among several options was taken, the values are not presented as optimums. No mathematical solution was undertaken as an attempt to find the best weighting factor, rather the selected controller is merely a functional and capable controller for the system.

4.4 Dynamic Case Studies

DESCRIPTION

In addition to the situation utilized in the above examples, the controllers and simulation were subjected to a series of other tests. First the simulation was tested to determine their response to a sinusiodal CO_2 production rate that always created a heavier load on the same sorbent bed. This would provide insight into the systems response at being excited at a given frequency. The results for this test can be found in figures 4.4-1 and 4.4-2. Here, as before,the upper curve is the partial pressure on the right axis, and the mass flow rate is the bottom curve scaled along the left hand axis.

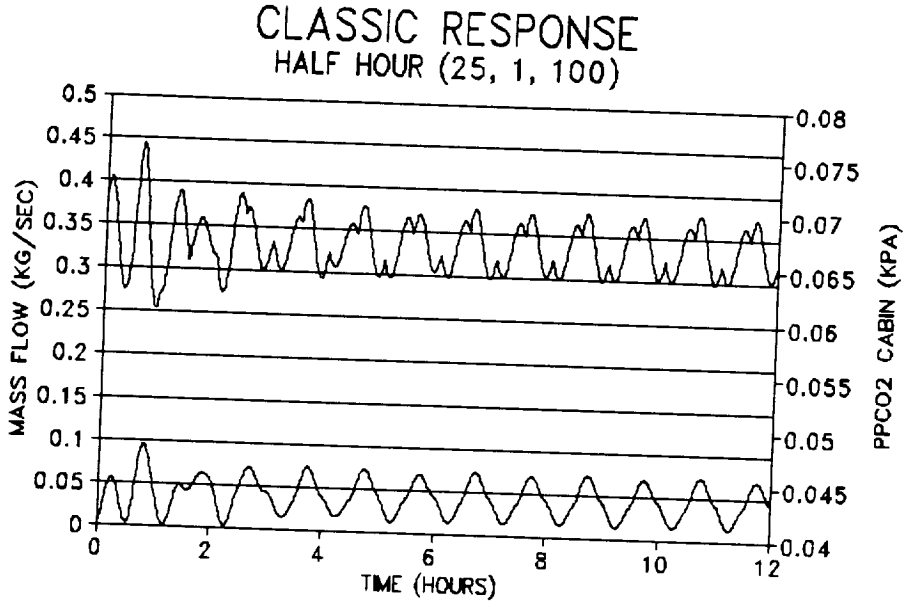

Figure 4.4-1 Classical Response to Half Hour Cycle

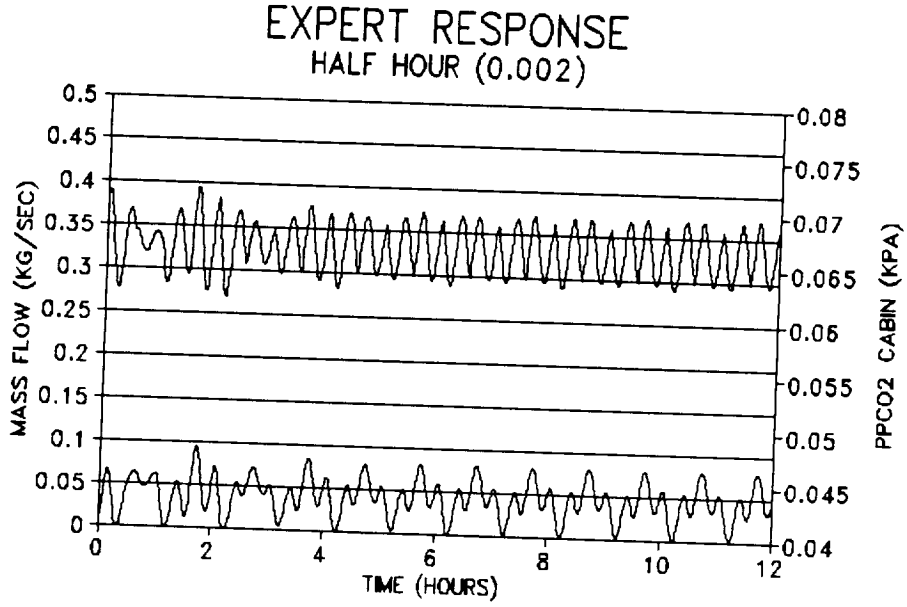

Figure 4.4-2 Expert Response to Half Hour Loading

The next case was conducted to determine the natural frequency of the controllers. By imparting a single impulse, in this case a short term high CO_2 production spike, it is possible to observe the systems natural frequency. The results of this test can be seen in Figures 4.4-3 and 4.4-4. It can be noticed that the expert controller has the higher frequency of the two. That does not necessarily imply that the expert controller has the faster response capability, only that it cycles as a higher rate. Also in this scenario it is easy to observe the dampening abilities of the control systems as they reduce the oscillations amplitudes. The final point of interest is the visibility of the half hour frequency imparted due to bed switching. It is what is responsible for the steady state oscillations visible in the graphs.

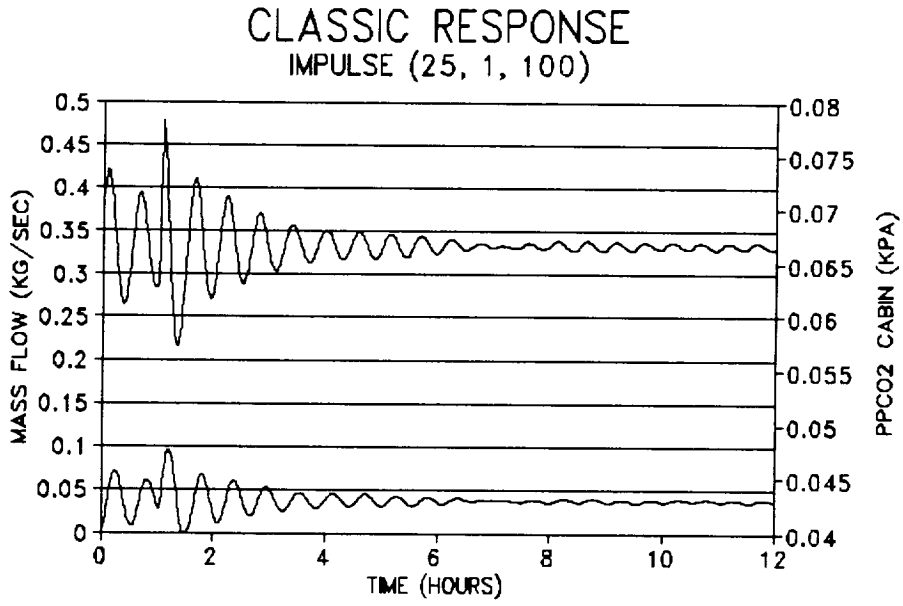

Figure 4.4-3 Classical Response to an Impulse

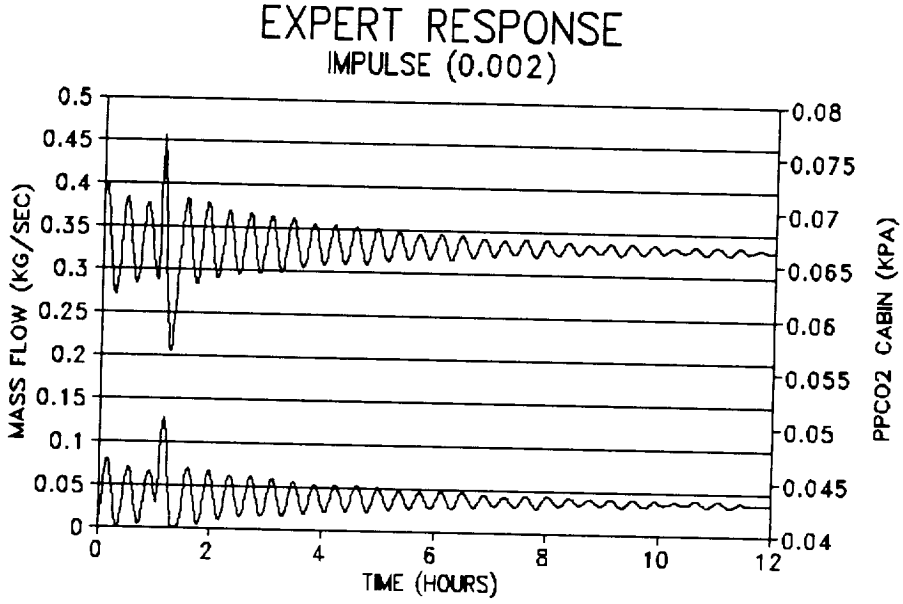

Figure 4.4-4 Expert Response to an Impulse

The final scenario examined was the controllers' abilty to handle a massive CO_2 production rate. This would simulate a fire in a space station module, or possibly a leak in the CO_2 accumulator tank. The results of this trial are given in Figures 4.4-5 and 4.4-6. The classical system was able to respond the quicker of the two, as evidenced by its more rapid increase of the mass flow rate. The slower response of the expert system results in the CO_2 partial pressure reaching a value of 14 kPa as opposed to the PID's peak value of 12 kPa. The major consideration however is how long before the CO_2 level returns to acceptable limits, and here the controllers both show the situation under control by 2 hours later.

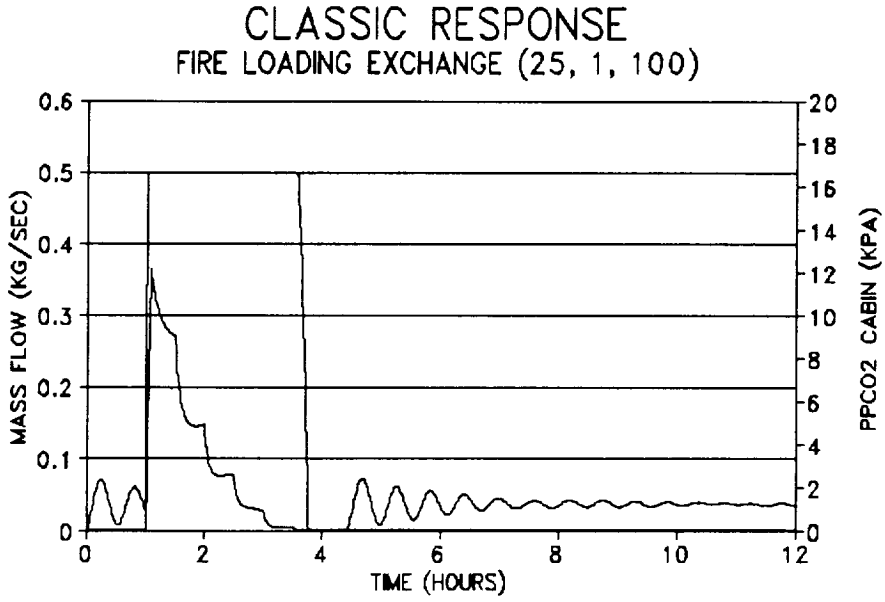

Figure 4.4-5 Classical Response to a Fire

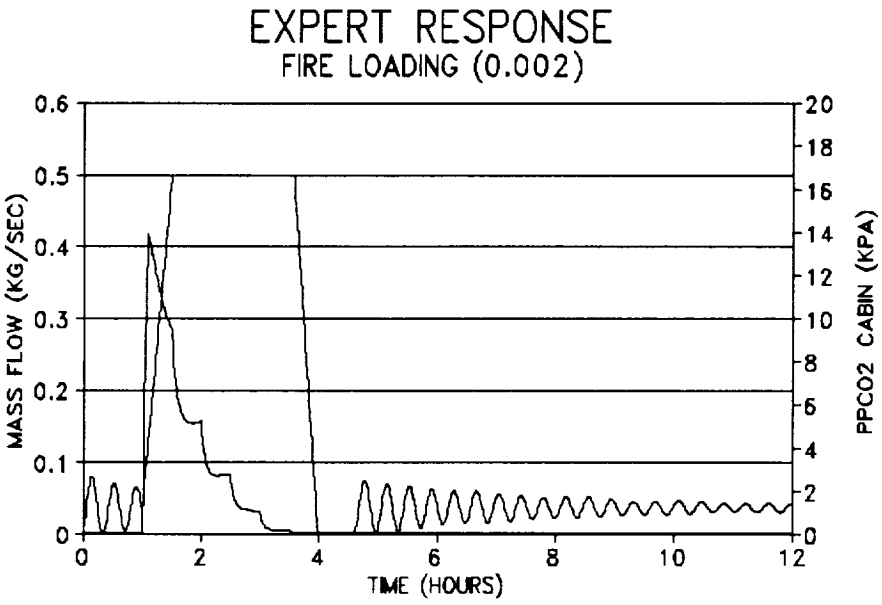

Figure 4.4-6 Expert Response to a Fire

4.5 Conclusions and Recommendations

CONCLUSIONS

The first conclusion that can be gathered from this report is that the simulation presented is a success. The physical phenomena modeled are accurate and respond correctly to parameter changes. This implies that the simulation is capable of being used as a test bed for evaluating almost any parameter's influence on the systems behavior. It is possible to determine the effects of the possible disasters (such as a fire), or to merely examine how the system operates under normal conditions.

Both controllers were found to be capable of handling the tasks assigned. There is currently no way to evaluate the controllers as far as superior capability. Neither was formally optimized, and so the limit of their abilities is still not known.

RECOMMENDATIONS

It is recommended that a formal optimization of the controllers be done. Once optimization is completed, a rigid and weighted set of criteria should be drafted. After testing the controls with the simulation code, the control schemes could be scored against the criteria. Once this is completed, the better control system should be implemented as the control scheme of choice. Note that a single type of control may not necessarily be the best choice. Rather, a control heirarchy where an expert system oversees a series of classical controls (or vice versa) might be the most effective choice.

57 INTENTIONALLY BLANK

5.0 APPENDICES

5.1 Modeling Lecture Summary

Dr. Byron Jones, professor of mechanical engineering at Kansas State University, introduced the design team students to the concepts of mathematical modeling and controls. The following is an outline of the lecture.

A. Definitions

1. Parameters. Parameters which are inherent to the system or which are determined from outside the system. Parameters do not change state variables but will affect how rate variables relate to state variables.

2. Inputs. Inputs are variables which change state variables. The values of Rate Variables are determined from outside the system.

3. System Relationships. The relationships that describe how the values of the rate variables are determined from the state variables. (This is the tough part.)

4. State Variables. Those variables required to define the state of a system. Specifying all of the state variables completely defines the state of a dynamic system. State variables do not change instantaneously.

5. Rate Variables. Rate variables are those which change state variables. The values of rate variables are determined by state variables.

B. Dynamic Systems Simulation

Development of a set of equations (or other model) that describes how a system behaves over time in response to various inputs to the system.

C. Dynamics Systems Analysis

Use of various techniques to study the dynamic behavior of a system (stability, speed of response, etc...)

D. System of Equations

A set of equations must be developed or applied to describe the behavior of the system being modeled, and they have the form

$$\frac{dX_1}{dt} = F_1(X_1, X_2, \ldots, X_n, Y_1, \ldots, Y_m) \quad (1)$$

E. Solving the Equations

1. Initial Conditions.
2. Calculate Derivatives.
3. Integrate One Time Step.
4. Calculate Derivatives.
5. Integrate One Time Step, (etc...).

F. Example of a Water Supply System

1. Verbal description of Components

a) State variables
L- water level (ft)
N- valve position (number of turns open)

b) Inputs
F_o- rate at which water is demanded (cfm)

c) Parameters
L_s- float level (10 ft)
W_m- maximum valve speed (10 rpm)
N_m- turns required to fully open valve (20 revolutions)
F_m- maximum water supply flow (100 cfm)
A - tank area (100 ft^2)

2. System equations

$$\frac{dL}{dt} = \frac{F_i}{A} - \frac{F_o}{A}, \quad (2)$$

$$\frac{dN}{dt} = W, \quad (3)$$

$$F_i = F_i \times N. \quad (4)$$

if $L < L_s$ and $N < N_m$ then

$$W=W_m. \tag{5}$$

if $L > L_s$ and $N > 0$ then

$$W=-W_m, \tag{6}$$

otherwise

$$W=0. \tag{7}$$

3. Initial Approximation

$$F_i=\frac{N}{N_m}\times F_m \tag{8}$$

4. Get rid of switch and replace with a P-D controller

a) Proportional controller

$$Output=K_1\times(L_{set}-L) \tag{9}$$

b) Derivative controller

$$Output=K_2\times\frac{dl}{dt} \tag{10}$$

c) Total output

$$Output_{total}=K_1\times(L_{set}-L)+K_2\times\frac{dL}{dt} \tag{11}$$

5.2 Expert Systems Lecture Summary

Harold Kraus, expert systems consultant at Kansas State University, gave a two hour lecture on expert systems. Harold developed an expert control model for the 1990-1991 NASA/USRA team. This model was developed using CLIPS, which has highly influenced his design and the design that is being implemented this year. The one problem with using expert systems however in this project is that it becomes increasingly difficult to model an Environmental Control System when the simulation is written in FORTRAN and the expert control is written in CLIPS. CLIPS is a closed system in that it cannot communicate with other programming languages except through files. Therefore, there comes a problem with passing variables between simulation and control modules. Not only is this difficult and time consuming to do, it causes the overall model to run slower because of the I/O between the disk and memory. To avoid this, Harold Kraus implemented the entire model within the CLIPS environment. This also made it difficult because anyone who has tried to implement a linear project in an expert systems language quickly learns that rules do not fire in the order that is expected.

In his lecture, Harold talks about the definitions, properties, applications, uses, and examples of an expert system. The following is an outline of the lecture:

I. Expert Systems
   - A. Definition of an expert system:
     An expert system is a system that contains expertise about knowledge. It makes decisions based on given information. The separations of the expert system lie in the knowledge base(facts and rules) and the engine(CLIPS) itself.
   - B. Properties of an expert system:
     Properties of an expert system should be consistency and reliability.
   - C. Applications and uses of an expert system:
     Applications of expert systems include classifications, diagnosis, design, and control. Expert systems should be used when there are many inputs and outputs and the relationship between those inputs and output are inexact or incomplete.
   - D. Examples of an expert system:
     One of the examples was the discussion of a fuzzy controller. Using a fuzzy controller gave a good example of a reasoning model and inexact reasoning. The model that was used was a simple float valve in a tank of water. The following is his example.

The initial conditions, as well as time step, run time limit, and model parameters are provided to the expert system through an initialization data file. Let the file define the following values:

```
TIME_STEP                                   = 0.2 minutes
WATER_LEVEL (position)                      = 10 feet
VALVE_POSITION                              = 0 revolutions
```

At t = 0 minutes,

```
LEVEL_ERROR                                 = 0 feet
WATER_LEVEL (rate)                          = 0 ft/min.
```

Applying the membership functions yields just one statement with a non-zero confidence factor:

```
LEVEL_ERROR is SMALL          confidence = 1.
```

Applying the operational rules yields the conclusion

```
MAINTAIN_VALVE_POSITION       confidence = 1.
```

The centroid of MAINTAIN_VALVE_POSITION is zero so

```
VALVE_SPEED                                 = 0 rpm.
```

Stepping the model through one time step using the calculated valve speed changes the input variables such as

at t = .2 minutes,

```
LEVEL_ERROR                                 = -0.1 feet
WATER_LEVEL                                 = -0.5 ft/min.
```

Applying the membership functions yields the following statements:

```
LEVEL_ERROR is POSITIVE       confidence = 0,
LEVEL_ERROR is SMALL          confidence = 0.8,
LEVEL_ERROR is NEGATIVE       confidence = 0.2,

WATER_LEVEL is FALLING        confidence = 1,
WATER_LEVEL is RISING         confidence = 0.
```

Notice how the confidence values add to 1. They are normalized so that we may obtain a percentage weight that gives a numeric confidence or belief that a fact is true. The higher the confidence number, the more we believe that the fact is true. The lower the confidence number, the less we believe that the fact is true.

Applying the operational rules yields the following conclusions:

OPEN_VALVE	confidence = 1,
CLOSE_VALVE	confidence = 0,
MAINTAIN_VALVE_POSITION	confidence = 0.8.

What could be done in this case is to compare the confidence values for OPEN_VALVE, MAINTAIN_VALVE_POSITION, and CLOSE_VALVE. The one with a higher confidence value would take precedence.

II. 1990-91 expert systems project

A. Explain what last years expert system controlled.

Last years problem dealt with the Oxygen Generation Assembly (OGA).

1. The sensory inputs into the Expert System included:
   a) C_2O
   b) H_2O
   c) O_2.
2. The command outputs included:
   a) Reduction of the amount of carbon dioxide
   b) Removal of carbon dioxide from the cabin.

B. Demonstrate the techniques used in a simple model

1. Triangular functions

A triangular function if one method of determining percent belief in a fact. With a triangular function, one determines the three points of the triangle by getting knowledge from an expert on the subject matter. Triangular functions are useful when trying to pull values within a predefined range. An example would be to keep the pressure inside a tank at a pressure P plus or minus some delta P.

2. Trapezoidal functions

A trapezoidal function is the same thing as a triangular function with the exception that it contains four points rather than three. This causes the function to take the form of a trapezoid with the top sometimes referred to as the plateau.

5.3 Homework 1 Summary

INTRODUCTION

The following is a summary of the first homework assignment presented to the advanced design team by Dr. Byron Jones, professor of Mechanical Engineering at Kansas State University. It's purpose was to introduce the design team students to the concepts of mathematical modeling.

PROJECT DESCRIPTION

Carbon dioxide is generated by a process at low pressure. A vane pump compresses the CO_2 into a storage tank where it is withdrawn periodically for use elsewhere. The pump runs at a constant speed (1000 rpm). The vane pump is volumetric, or it pumps a fixed volume (40 cm^3) of CO_2 into the tank each revolution. The tank is perfectly insulated so there is no heat loss to the ambient environment. Maximum allowable tank pressure is 300 kPa.

INITIAL CONDITIONS

Case 1. There is no usage of the stored CO_2 and the inlet temperature and pressure remain constant at 40 °C and 25 kPa respectively.

Case 2. Same as case 1 except the inlet pressure decreases with time according to the relation below

$$P = P_o \times e^{(-t/a)}. \qquad (12)$$

Stop the simulation after 60 minutes if the limits above are not reached prior to that time.

The symbols used in the above equation are defined as follows.

P = pressure as a function of time,
P_o = initial pressure (25 kPa),
t = time from start up (minutes),
a = a constant (10 minutes).

Case 3. Same as case 2 except now CO_2 is used from the reservoir at the rate of 0.3 g/s starting at a time 2 minutes and continuing for 5 minutes.

ASSIGNMENT

Develop a dynamic simulation model of the behavior of the tank pump system for all three cases.

DYNAMIC MODEL

1. Verbal Description of Components

   a) State Variables
   P - Tank Pressure (kPa)
   T - Tank Temperature ($^{\circ}C$)

   b) Inputs
   M_{out} - rate that CO_2 is demanded
   P_i - inlet pressure to the pump as a function of time

   c) Parameters
   W_p - speed of pump (1000 rpm)
   V_p - displacement of volumetric pump (40 cm^3)
   $P_{o,tank}$ - initial pressure of tank(25 kPa)
   $T_{o,tank}$ - initial temperature of tank(25$^{\circ}C$)
   V_t - Volume of tank (.25 m^3)
   $P_{i,initial}$ - initial pressure at inlet of pump (25 kPa)
   $T_{i,initial}$ - initial temperature at inlet of pump (40$^{\circ}C$)

2. System Equations

$$\underline{U}_{tank}(\underline{t}) = \underline{C}_v \times \underline{M}_{tank}(\underline{t}) \times (\underline{T}_{tank}(\underline{t}) - 273) \quad (1)$$

$$\underline{M}_{tank}(\underline{t}+\underline{dt}) = \underline{M}_{tank}(\underline{t}) + \frac{\partial \underline{M}_{tank}}{\partial \underline{t}} \times \underline{dt} \quad (2)$$

$$\underline{U}_{tank}(\underline{t}+\underline{dt}) = \underline{U}_{tank}(\underline{t}) + \frac{\partial \underline{H}}{\partial \underline{t}} \times \underline{dt} \quad (3)$$

$$\underline{T}_{tank}(\underline{t}+\underline{dt}) = \frac{\underline{U}_{tank}(\underline{t}+\underline{dt})}{\underline{M}_{tank}(\underline{t}+\underline{dt}) \times \underline{C}_v} + 273 \quad (4)$$

$$P_{tank}(t+dt) = \frac{R \times T_{tank}(t+dt) \times M_{tank}(t+dt)}{V_{tank}} \qquad (5)$$

3. Simulation (Computer Program)

```
      PROGRAM PUMPTANK
      REAL MPIN,MPUMP,MTANK,MUSED

C ... INITIALIZE VARIABLES
      SPEED= 1000.
      RCO2=0.1889
      VPUMP=0.00004
      VTANK=0.25
      CVCO2=0.657
      CPCO2=CVCO2+RCO2
      KCO2=CPCO2/CVCO2
      H0=RCO2*273
      TIME=0.
      TTANK=25+273
      PTANK=25.
      PPIN=25.
      TPIN=40.+273.
      MTANK=VTANK*PTANK/(RCO2*TTANK)
      UTANK=(TTANK-273)*CVCO2*MTANK
      MUSED=0.0
      DTIME=.1
      TTANK=UTANK/(MTANK*CVCO2)+273
      PTANK=RCO2*TTANK*MTANK/VTANK
      OPEN(UNIT=11,FILE='HW1.OUT')
      WRITE(11,*)'TIME(MIN) TEMP (C)     PRESS. (kPa)   MASS,TANK'

C ... CALCULATE CURRENT STATES
      TO=TIME
1     IF((TIME-TO+.05).GT.1.)THEN
        WRITE(11,100)TIME,TTANK-273,PTANK,MTANK
        TO=TIME
      END IF

C ... UNCOMMENT FOR CASE 3
      IF(TIME.GT.1.95.AND.TIME.LT.7.05)THEN
         MUSED= .0003
      ELSE
         MUSED= 0.
      END IF
      WRITE(*,*) TIME,MUSED
C ...
      MPUMP=SPEED*VPUMP*PPIN/(RCO2*TPIN)
      TPUMP=TPIN*(PTANK/PPIN)**(1-1/KCO2)
      HIN=MPUMP*((TPUMP-273)*CPCO2+H0)
```

```
      HOUT=MUSED*((TTANK-273)*CPCO2+H0)
      TIME=TIME+DTIME
      MTANK=MTANK+(MPUMP-MUSED)*DTIME
      UTANK=UTANK+(HIN-HOUT)*DTIME
      TTANK=UTANK/(MTANK*CVCO2)+273
      PTANK=RCO2*TTANK*MTANK/VTANK

C ... UNCOMMENT FOR CASE 2 AND CASE 3
      PPIN=25.*EXP(-TIME/10)
      IF(TIME.GT.60)GOTO 2
C ...
      IF(PTANK.LT.300..AND.TTANK.LT.(250+273))GOTO 1
2     WRITE(11,100)TIME,TTANK-273,PTANK,MTANK
100   FORMAT(F5.2,3(2X,F10.4))
      END
```

4. Results

Case 1. The conditions for the first case described a constant inlet temperature and pressure. Figure 5.3-1 shows results of the first case. Notice that because a constant inlet pressure is maintained, the pressure within the tank increases linearly. As the pressure increased with time, the temperature seemed to approach an asymptotical value of about 394 K. The simulation was halted after 53 minutes due to the pressure exceeding its limit.

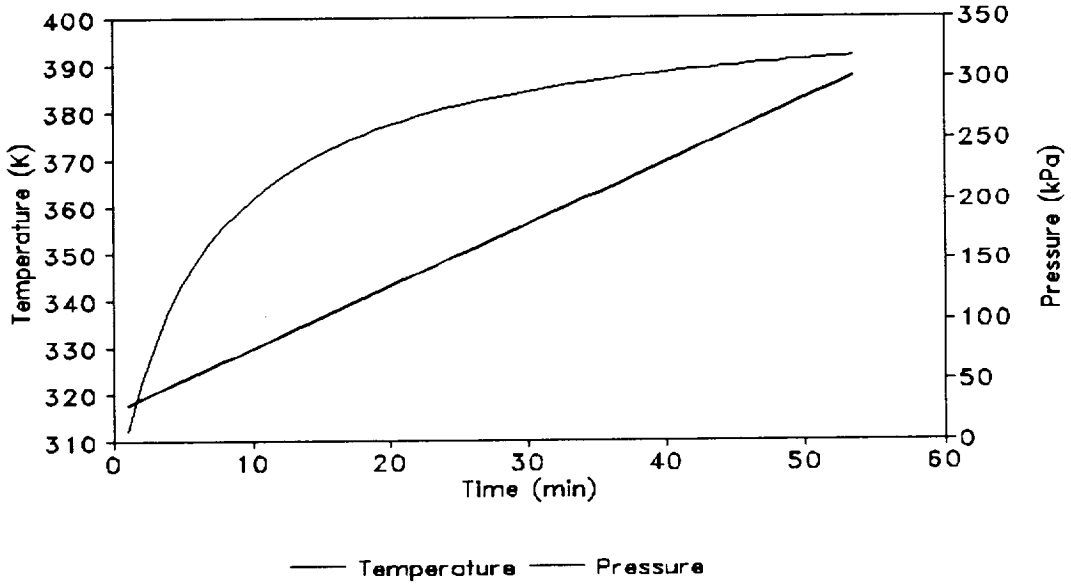

Figure 5.3-1: Homework 1 Case 1 Data

Case 2. Case 2 was the same as case 1 except for a decreasing inlet pressure according to a given function. As shown in Figure 5.3-2, the temperature and pressure seemed to level out at 360 K and 77 kPa respectively. The pressure within the tank responded as would be anticipated with an exponentially decreasing input function for the inlet pressure. The temperature increased much more rapidly for case 2 from 0 to 10 minutes, and it leveled out at a values about 30 K less than the previous simulation. No limits on the tank were exceeded so the simulation automatically halted at 60 minutes.

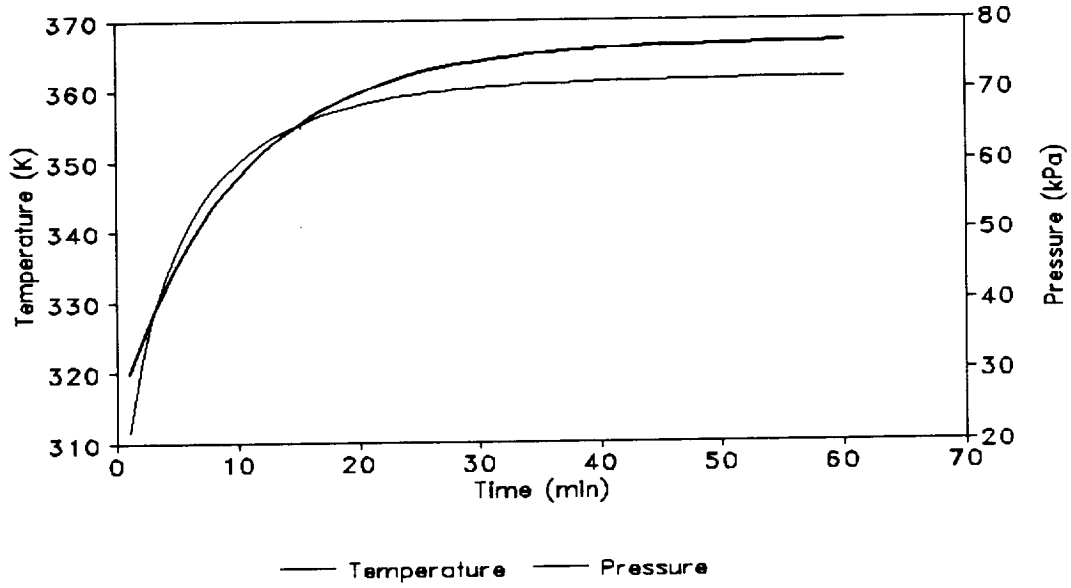

Figure 5.3-2: Homework 1 Case 2 Data

Case 3. Case 3 added a withdrawal of .3 g/s from the tank from 2 minutes to 7 minutes. The results from this simulation, shown in Figure 5.3-3, were similar to that of Case 2, and the effects of the new condition seemed negligible. If figures for Cases 2 and 3 were examined closely from 2 minutes to 7 minutes, slightly less slope would be seen in both responses due to the drain on the tank.

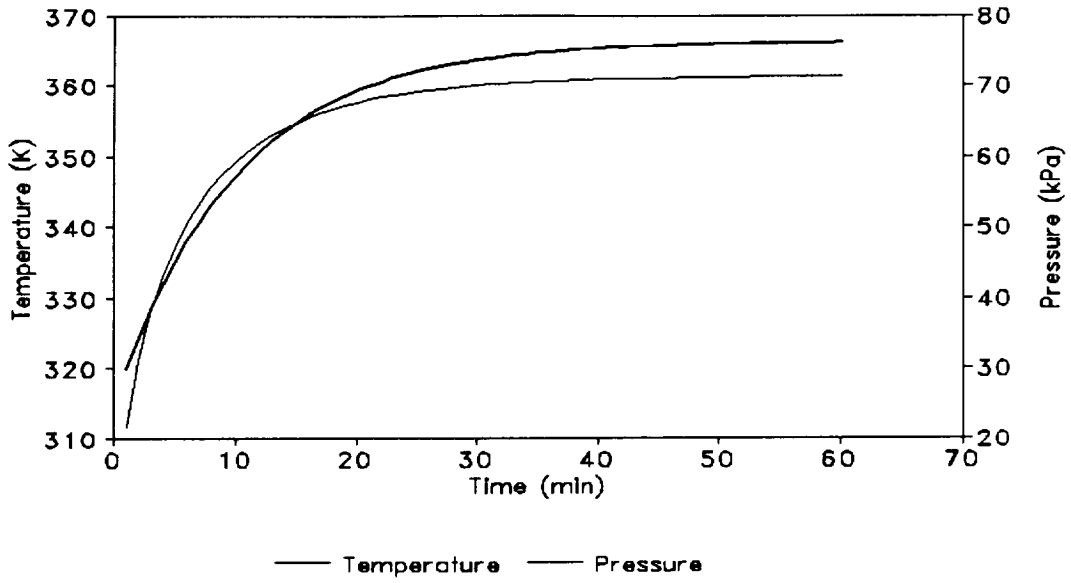

Figure 5.3-3: Homework 1 Case 3 Data

5.4 Homework 2 Summary

INTRODUCTION

The following is a summary of the second homework assignment presented to the advanced design team. It's purpose was to introduce the design team students to the concepts of simulation diagrams and numerical integration of differential state equations using methods other than the Euler time step.

PROJECT DESCRIPTION

The system examined is shown in Figure 5.4-1. It is a simplified automobile suspension consisting of ¼ of the automobile mass, M_1, the suspension system mass, M_2, the suspension spring, K_1, the elasticity of the tire, K_2, and the shock absorber, B. The vertical displacement of the road is input to the system by a force, f(t), acting on the suspension mass. A major simplification is made by assuming the tire to never leave the road surface. The coefficients and forcing function are given as follows.

$M_1 = 250$ kg
$M_2 = 30$ kg
$B = 1.5$ kg/s

$K_1 = .55$ kg/s^2
$K_2 = 20$ kg/s^2
$f(t) = 10\, e^{-t} \sin(2\pi t/300)$ N

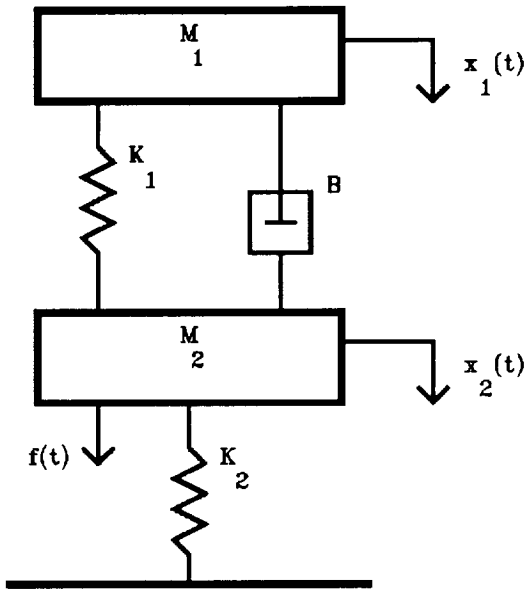

Figure 5.4-1: Simplified Automobile Suspension System

ASSIGNMENT

1. Write dynamic equations that simulate this system. Summing forces on each mass should result in two second-order differential equations.

2. Draw a simulation diagram from these simulation equations. How many integrators should it have?

3. Write a set of state equations describing this system.

4. Produce a set of four plots: M_1 position, M_2 position, M_1 velocity, and M_2 velocity. To produce these plots, write or use a computer program to perform the simulation.

SOLUTION

1. Summing forces on M_1 and M_2 gives equations 1 and 2.

$$M_1 \frac{d^2 x_1}{dt^2} = -B\left(\frac{dx_1}{dt} - \frac{dx_2}{dt}\right) - K_1(x_1 - x_2) \qquad (1)$$

$$M_2 \frac{d^2 x_2}{dt^2} = f(t) - B\left(\frac{dx_2}{dt} - \frac{dx_1}{dt}\right) - K_1(x_2 - x_1) - K_2 x_2 \qquad (2)$$

2. Figure 4.5-2 is the simulation diagram with four integrators.

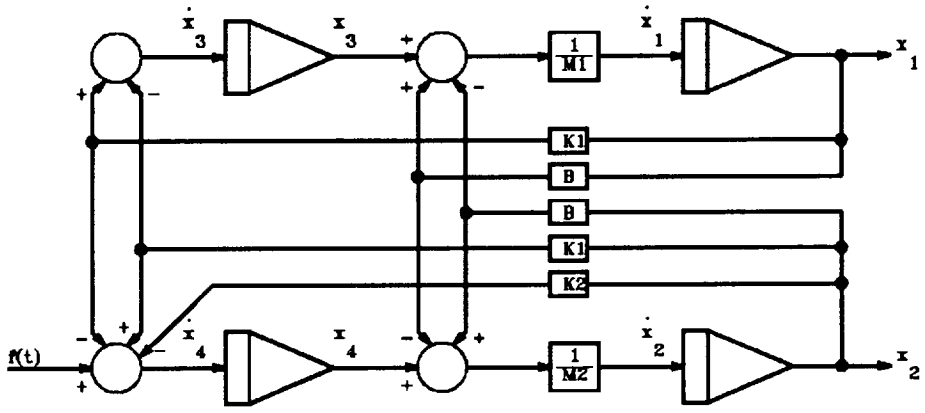

Figure 5.4-2: Homework 2 Simulation Diagram

3. The following 3-6 are a set of state equations obtained directly from the simulation diagram.

$$\dot{\underline{x}}_1 = \frac{1}{\underline{M}_1}\left(-\underline{B}\underline{x}_1 + \underline{B}\underline{x}_2 + \underline{x}_3\right), \tag{3}$$

$$\dot{\underline{x}}_2 = \frac{1}{\underline{M}_2}\left(\underline{B}\underline{x}_1 - \underline{B}\underline{x}_2 \quad \underline{x}_4\right), \tag{4}$$

$$\dot{\underline{x}}_3 = -\underline{K}_1\underline{x}_1 + \underline{K}_1\underline{x}_2, \tag{5}$$

$$\dot{\underline{x}}_4 = \underline{f}(\underline{t}) - \underline{K}_1\underline{x}_2 + \underline{K}_1\underline{x}_1 - \underline{K}_2\underline{x}_2. \tag{6}$$

Another set of state equations is arrived at by first making the definitions

$$\dot{\underline{x}}_1 = \underline{x}_3, \tag{7}$$

$$\dot{\underline{x}}_2 = \underline{x}_4. \tag{8}$$

Substituting these into the second-order differential Equations 1 and 2 gives

$$\dot{\underline{x}}_3 = \frac{1}{\underline{M}_1}\left[-\underline{B}\left(\underline{x}_3 - \underline{x}_4\right) - \underline{K}_1\left(\underline{x}_1 - \underline{x}_2\right)\right], \tag{9}$$

$$\dot{\underline{x}}_4 = \frac{1}{\underline{M}_2}\left[\underline{f}(\underline{t}) - \underline{B}\left(\underline{x}_4 - \underline{x}_3\right) - \underline{K}_1\left(\underline{x}_2 - \underline{x}_1\right) - \underline{K}_2\underline{x}_2\right]. \tag{10}$$

4. Plots of M_1 position, M_2 position, M_1 velocity, and M_2 velocity can be seen in Figures 5.4-3 through 5.4-6. The solution was arrived at by solving the equations every .1 seconds for 300 seconds using the $4^{th}/5^{th}$ order Runga-Kutta-Fehlburg method

(RKF45 subroutine). The results are underdamped solutions. The "bump" causes the suspension M_2 to vibrate for roughly 4 minutes, while the automobile M_1 has only lost half of its vibrational amplitude by the end of the 5 minute simulation.

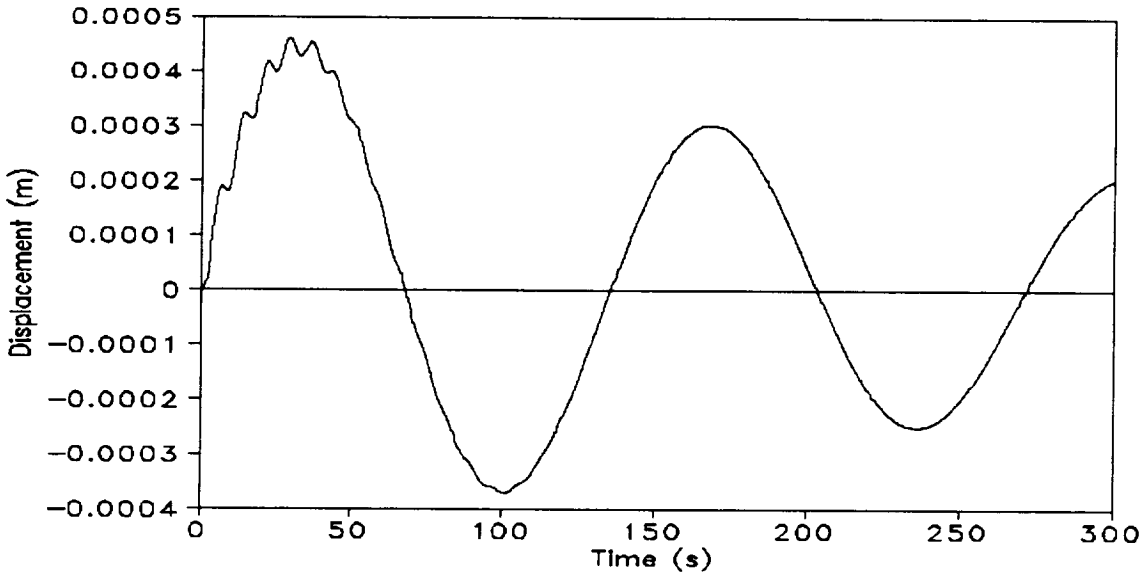

Figure 5.4-3: M_1 Position

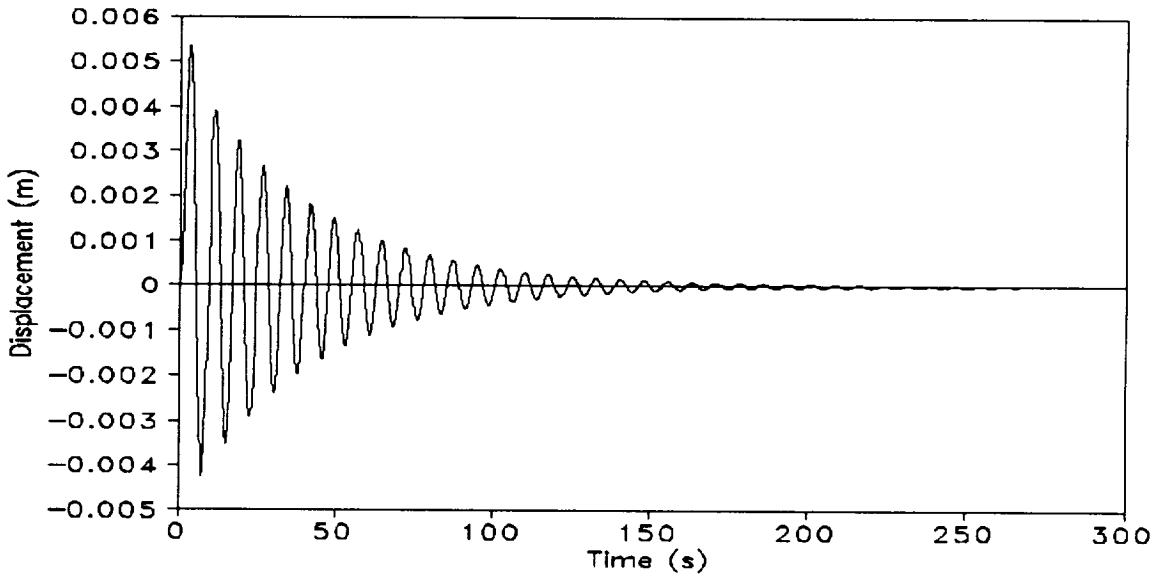

Figure 5.4-4: M_2 Position

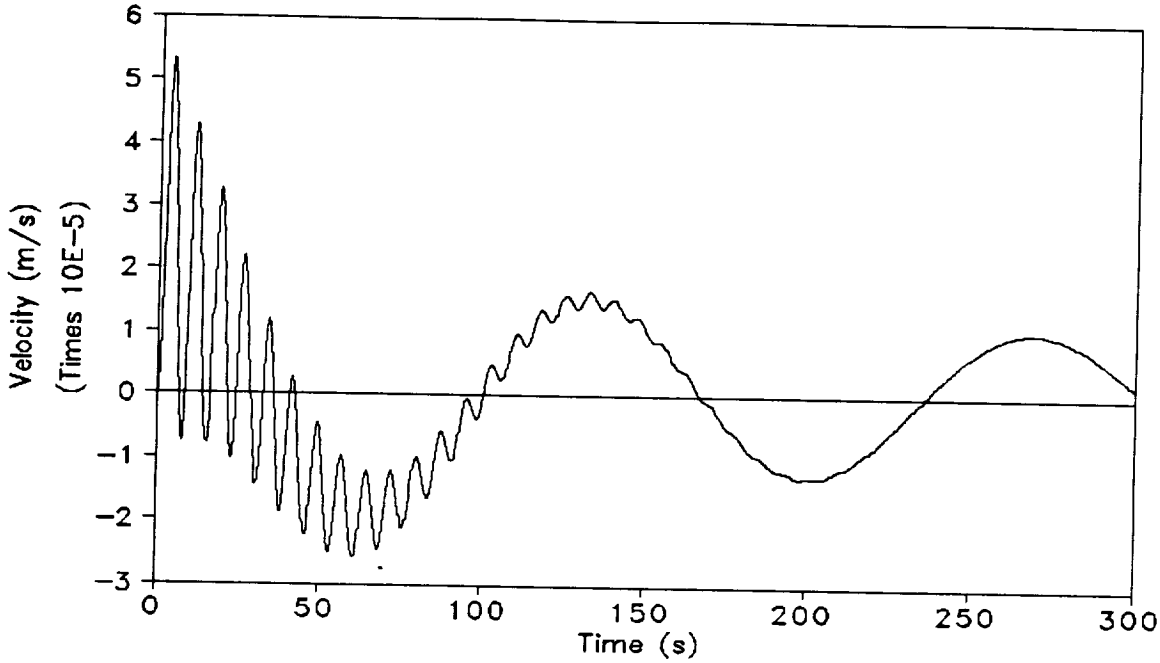

Figure 5.4-5: M_1 Velocity

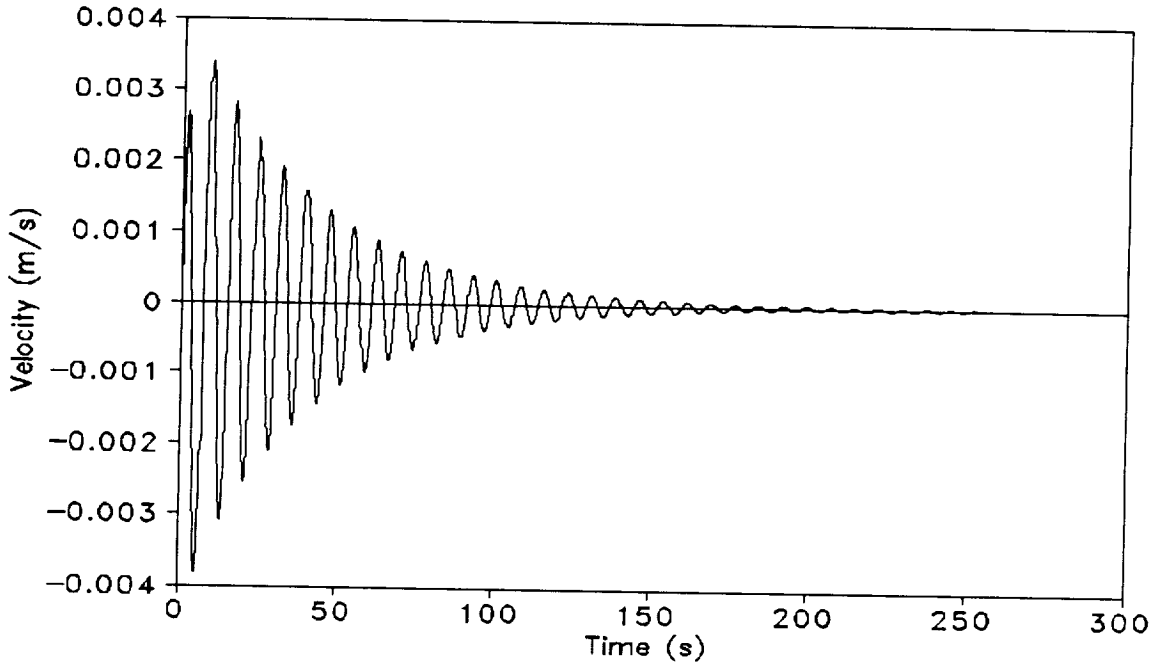

Figure 5.4-6: M_2 Velocity

Computer Program

```
C$INCLUDE DEQ
C$INCLUDE RKF45

      PROGRAM HW1
      INTEGER IWORK(5)
      DOUBLE PRECISION X(4),T,TOUT,RELERR,ABSERR,WORK(27)
      EXTERNAL DEQ

      N=4
      T=TOUT=0.
      X(1)=X(2)=X(3)=X(4)=0.
      IFLAG=1
      RELERR=1E-08
      ABSERR=0.
      OPEN(UNIT=2,FILE='HW1.DAT')
100   FORMAT(F6.1,4E12.4,I3)
      WRITE(2,100) T,X(1),X(2),X(3),X(4),IFLAG
      DO10I=1,3000
        TOUT=I/10.
5       CALL RKF45(DEQ,N,X,T,TOUT,RELERR,ABSERR,IFLAG,WORK,IWORK)
        IF(IFLAG.NE.2) THEN
          WRITE(6,'(A,I2,A,F6.1,A)') 'IFLAG=',IFLAG,' at',T,
     &       ' seconds...trying again'
          IF(IFLAG.EQ.7) IFLAG=2
          GOTO5
        ENDIF
        WRITE(2,100) T,X(1),X(2),X(1),X(2),IFLAG
10    CONTINUE
      END

      SUBROUTINE DEQ(T,X,XDOT)
      DOUBLE PRECISION T,X(*),XDOT(*),F
      REAL B,K1,K2,M1,M2

      B=1.5
      K1=.55
      K2=20
      M1=250
      M2=30
      F=10*EXP(-T)*SIN(T*2*ACOS(-1.)/300)
      XDOT(1)=X(3)
      XDOT(2)=X(4)
      XDOT(3)=(B*(X(4)-X(3))+K1*(X(2)-X(1)))/M1
      XDOT(4)=(B*(X(3)-X(4))+K1*(X(1)-X(2))-K2*X(2)+F)/M2
C      XDOT(1)=(B*(X(2)-X(1))+X(3))/M1
C      XDOT(2)=(B*(X(1)-X(2))+X(4))/M2
C      XDOT(3)=K1*(X(2)-X(1))
C      XDOT(4)=K1*(X(1)-X(2))-K2*X(2)+F
      RETURN
      END
```

5.5 Classical Controls Source Code

```
C$NOEXT
C$NOWARN
C*****************************************************************
C      PROGRAM NAME:    CCONTROL.FOR
C      GROUP NAME:      CLASSICAL CONTROLS
C      CREATED BY:      STAN KROEKER
C      REVISED:         02MAR92  -  CREATED
C                       04MAR92  -  COMPLETED FIRST TIME
C                       16MAR92  -  ADDED SYSTEM CONTROL AND CABIN
C                                   SUB.
C                       17MAR92  -  ADDED VALVE BETWEEN PUMP AND
C                                   TANK
C                       22MAR92  -  ADDED BOSCH REACTOR
C                       24MAR92  -  TESTED ALL CONTROLS AND REMOVED
C                                   CABIN.FOR - SATISFIED WITH PROGRAM
C
C      CALLS:           READS CDATA
C      CALLED FROM:     C.FOR
C      OUTPUT FILE:     WRITES TO CDATA
C
C    LIST OF VARIABLES:
C                .
C    INT      TIME = CURRENT TIME
C    REAL     MDOT = MASS FLOW OF AIR INTO SYSTEM IN Kg/SEC
C    REAL     PCO2 = PARTIAL PRESSURE OF CO2 IN CABIN AIR
C    INT      BLOSPEED = BLOWER SPEED, ONE OF THREE
C                     1 -- NORMAL
C                     2 -- DEGRADED
C                     3 -- EMERGENCY
C    INT      BEDSWITCH = TIME INTERVAL FOR SWITCHING SORBENT BEDS
C                         -- SEC
C    REAL     PTANK = C02 ACCUMULATOR TANK PRESSURE -- KPa
C    REAL     PUMPSPEED = C02 ACCUMULATOR TANK PUMP SPEED -- RPM
C    REAL     PTANK_SET = DESIRED PRESSURE OF CO2 TANK -- KPa
C    REAL     PTANK_OLD = CO2 TANK PRESSURE FROM LAST TIME STEP
C                         -- KPa
C    INT      SYS       = SYSTEM STATUS, 1 = ON, 0 = OFF
C    INT      SYSTIME   = TIME SINCE SYSTEM WAS LAST TURNED ON,
C                         VARIES FROM 0-1800
C    REAL     CO2REM    = CO2 REMOVED BY SYSTEM
C                            IF VALVE = 0, CO2REM = MCO2R
C                            IF VALVE = 1, CO2REM = 0
C    REAL     PB        = PRESS OF DESORBING SORBENT BED -- KPa
C    REAL     PBCO2     = EQUILIBRIUM PRESS OF CO2 IN DESORBING
C                            SORBENT BED -- KPa
C    REAL     MAIR      = MASS OF AIR LEAVING DESORBING SORBENT
C                            BED -- Kg/SEC
C    REAL     MCO2R     = MASS OF CO2 LEAVING DESORBING SORBENT
C                            BED -- Kg/SEC
```

```
C    REAL     MSBOUT     = MASS OF AIR LEAVING ADSORBING SORBENT
C                              BED -- Kg/SEC
C    INT      VALVE      = STATE OF VALVE BETWEEN PUMP AND TANK
C                              1 DIRECTS GAS TO CABIN
C                              0 DIRECTS GAS TO TANK
C    REAL     MTANKUSED = MASS OF CO2 USED BY BOSCH REACTOR  Kg/SEC
C
C    DESCRIPTION:
C
C    PROGRAM READS FROM CDATA TO DETERMINT THE CURRENT STATUS OF
C    THE CONTROL PARAMETERS, CALCULATIONS ARE THEN MADE TO CHANGE
C    THE CONTROL VARIABLE IF NEEDED.
C    TO CONTROL THE PARTIAL PRESSURE OF CO2 IN THE CABIN:
C       TURNS THE SYSTEM ON FOR 1800 SEC IF THE PCO2 > .4KPa
C    TO CONTROL THE CO2 ACCUMULATOR TANK PRESSURE:
C       CHANGE THE CO2 ACCUMULATOR PUMP SPEED
C
C*****************************************************************

      SUBROUTINE PID(SIMTIME,DT,MDOT,PRESS,TEMP,PHI,TIME,PCO2,
     &           BLOSPEED,SYSTIME2,PTANK,PUMPSPEED,PTANK_SET,
     &           PTANK_OLD,SYS,SYSTIME,CO2REM,PB,PBCO2,MCO2R,
     &           MCO2P,VALVE,MTANKUSED)

        REAL PTANK,PUMPSPEED,PTANK_SET,PTANK_OLD,PRESS
        DOUBLE PRECISION MDOT,MCO2R,MCO2P
        REAL PBCO2
        DOUBLE PRECISION MTANKUSED,CO2REM
        INTEGER TIME,BLOSPEED,BEDSWITCH,SIMTIME,DT,SYS,SYSTIME
        INTEGER VALVE,SYSTIME2

C ...   PID CONTROL OF CO2 REMOVAL SYSTEM, BASED ON THE PARTIAL
C       PRESSURE CO2 IN THE CABIN.
      SYS = 1
      IF(TIME.EQ.0)PCO2_OLD=PCO2
      PCO2_SET = 0.0667
      K1 = 25.
      K2 = 1.
      K3 = 100.
      ERR = PCO2-PCO2_SET
      DERR = PCO2-PCO2_OLD
      ERRDER = DERR/DT
      ERRINT=DT*(ERR+(PCO2_OLD-PCO2_SET))/2
      DMDOT=(ERR/K1+ERRDER/K2+ERRINT/K3)
      MDOT = MDOT + DMDOT
         IF(MDOT.GE.0.5)MDOT = 0.5
         IF(MDOT.LE.1E-3)MDOT = 1E-3
      PCO2_OLD = PCO2


         SYSTIME2=SYSTIME2+DT
         SYSTIME=SYSTIME+DT
```

```
C ...  CONTROL OF VALVE BETWEEN CO2 PUMP AND CO2 TANK
      IF((PB-PBCO2).GT.(.01*PBCO2)) THEN
         PUMPSPEED = 3000
         VALVE = 1
         CO2REM = MCO2R-MCO2P
         GOTO 200
       ELSE
         VALVE = 0
         CO2REM = MCO2R
         PUMPSPEED = 0
      ENDIF

200   CONTINUE

C ...  MASS FLOW DEMAND OF THE BOSCH REACTOR

      IF(PTANK.GE.101.125) THEN
         MTANKUSED = 4.633333E-5
       ELSE
         MTANKUSED = 0.
      ENDIF

      RETURN
      END
```

5.6 Expert Systems Source Code

```
;-------------------------------------------------------------------
;           EXPERT CONTROLS FOR NASA DESIGN PROJECT
;-------------------------------------------------------------------
;    Created          : February 21, 1992
;    Programmers      : Michael W. Honas
;                     : Robert A. Swenson
;    With help by     : Dr. David A. Gustafson
;                     : Stan Kroeker
;    Code             : ~/nasa/mike/expert.clp
;    Source Machine   : DEPOT
;    Modified         : April 3, 1992 -removed unused variables and
;                                      modified the controls
;                                      combined VALVE-TO-TANK & VALVE-TO-CABIN
;                                      into VALVE-TO-CABIN
;
;
;    Description:
;
;         This will monitor the simulation and write
;    new control parameters to the 'rates' file.  The only
;    expert controlled device this far are the tank pressure
;    and the cabin pressure.  Other devices being controlled
;    by on/off switches are as follows: CO2, valve to the cabin,
;    and bosch flow rate.
;
;-------------------------------------------------------------------

;-------------------------------------------------------------------
;               TRIANGULAR FUNCTIONS
;
;    The following fuzzy sets manually define the two triangles that
;    will determine if the pressure in the tank/cabin is too high
;    too low, or anywhere inbetween.  The beauty of this method is
;    that these triangles can easily be changed to tweak the results.
;    Further more, this can be done on the fly while the program is
;    running.  Simply edit the file and then change the values and
;    save them back out.
;
;-------------------------------------------------------------------

(deffacts start
        (state openedata)
        (fuzzy tank-pressure    low -150 300 400)
        (fuzzy tank-pressure    high 300 400 650)
     (fuzzy cabin-pressure    low  -100 .058 .075)
     (fuzzy cabin-pressure    high .058 .075  100)
)
```

```
;-------------------------------------------------------------------
;                    OPEN-edata
;
;    Read in the values from the Fortran and C simulation modules.
;    "edata" is a temporary file that is used to pass variables
;    between the simulation and the clips control.  This is done
;    simply because we cannot link CLIPS in with any other compiler.
;    This is one of the few draw backs is that it is almost a closed
;    system.
;
;-------------------------------------------------------------------

(defrule open-edata
     ?s <- (state openedata)
=>
;    (printout t "Entering open expert data" crlf)
     (retract ?s)
     (open "edata" data "r+")
          (assert (time-step      =(read data)))
          (assert (mass-in        =(read data)))
          (assert (partial-co2    =(read data)))
          (assert (tank-pressure   =(read data)))
          (assert (pump-speed =(read data)))
          (assert (sys        =(read data)))
          (assert (sys-time   =(read data)))
          (assert (sys-2time  =(read data)))
          (assert (co2-rem    =(read data)))
          (assert (pb              =(read data)))
          (assert (pbco2           =(read data)))
          (assert (mco2p      =(read data)))
          (assert (mco2r      =(read data)))
          (assert (valve      =(read data)))
          (assert (mtankused  =(read data)))

     (close data)
     (assert (state control-pp))
)

;-------------------------------------------------------------------
;                    CONTROL-PCO2
;
;    This rule controls the partial pressure of the carbon dioxide
;    and also takes care of the system time.
;
;-------------------------------------------------------------------

(defrule control-pco2
        ?s <- (state control-pp)
        ?f <- (sys ?sy)
        ?h <- (sys-time ?st)
        ?i <- (sys-2time ?s2t)
        ?j <- (mass-in ?mi)
```

```
        (partial-co2 ?pc)
        (time-step ?ts)

=>
;       (printout t "Entering CONTROL-PCO2" crlf)
        (retract ?s)

        (if (= ?sy 1) then
                (retract ?i)
                (assert (sys-2time =(+ ?s2t ?ts)))
        )

        (if (and (and (= ?sy 1) (>= ?st 1800))(<= ?pc .4)) then
                (retract ?f)
                (retract ?h)
                (retract ?j)
                (assert (sys 0))
                (assert (sys-time 0))
                (assert (mass-in 0))
        else
        (if (and (= ?sy 0) (>= ?pc .4)) then
                (retract ?f)
                (retract ?j)
                (assert (sys 1))
                (assert (sys-time 0))
                (assert (mass-in .5))
        else
        (if (and (and (= ?sy 1)(>= ?st 1800))(> ?pc .4)) then
                (retract ?h)
                (assert (sys-time 0))
                (assert (state valve))
        else
        (if (and (= ?sy 1)(< ?st 1800)) then
                (retract ?h)
                (assert (sys-time =(+ ?ts ?st)))
        ))))

        (assert (state valve))
)

;-------------------------------------------------------------------------
;                    VALVE-TO-CABIN
;
;    This system will turn the valve on and off and assert a
;    pump speed of 3000 if necessary.
;
;
;-------------------------------------------------------------------------

(defrule valve-to-cabin
     ?s <- (state valve)
     (pb ?pb)
```

```
      (pbco2 ?p2)
      ?f <- (pump-speed ?ps)
      ?g <- (valve ?va)
      ?h <- (co2-rem ?co)
      (mco2p ?mcp)
      (mco2r ?mcr)

=>
;     (printout t "Entering VALVE-TO-CABIN" crlf)
      (retract ?s)
      (retract ?g)
      (retract ?h)

      (if (> (- ?pb ?p2) (* ?p2 0.01))    ; if pb >> p2
      then
           (retract ?f)
           (assert (pump-speed 3000))
           (assert (valve 1))
           (assert (co2-rem =(- ?mcr ?mcp)))
           (assert (state bosch))
      else
           (assert (valve 0))
           (assert (co2-rem ?mcr))
           (assert (state calc))
      )
)

;---------------------------------------------------------------------
;               CALC-MEMBERSHIP-FOR-PUMP
;
;     Calculate the percent belief that the tank pressure or cabin
;     pressure is low, right or high.
;
;---------------------------------------------------------------------

(defrule calc-membership
      (state calc)
      (fuzzy ?var ?qual ?low ?mid ?high)
      (?var ?val)
=>

;     (printout t "Entering CALC-MEMBERSHIP-FOR-PUMP" crlf)
      (if (and (> ?val ?low) (<= ?val ?mid)) then
           (assert (member ?var ?qual =(/ (- ?val ?low) (- ?mid ?low)))))
      (if (and (> ?val ?mid) (< ?val ?high)) then
           (assert (member ?var ?qual =(/ (- ?high ?val) (- ?high ?mid)))))
)

;---------------------------------------------------------------------
;                    RULES
;
;     This rule decides whether an action should take place.
```

```
;    That action is based on the belief of better than 50%
;
;---------------------------------------------------------------------

(defrule rules
     (state calc)
     ?f <- (member ?variable ?quality ?val&:(> ?val .5))
=>
;    (printout t "Entering RULES" crlf)

     (retract ?f)
     (assert (action ?variable ?quality ?val))
)

;---------------------------------------------------------------------
;                   NO-CHANGES
;
;    Conversly, this rule decides that no action should take place
;    if the belief is 50% or less.
;
;---------------------------------------------------------------------

(defrule no-changes
     (state calc)
     ?f <- (member ?variable ?quality ?val&:(<= ?val .5))
=>
;    (printout t "Entering NO-CHANGES" crlf)

     (retract ?f)
)

;---------------------------------------------------------------------
;                   ACTION-SLOW-MOTOR
;
;    This rule will slow the motor by some variable amount
;    based on the percent belief.
;
;---------------------------------------------------------------------

(defrule action-slow-motor
     (state calc)
     ?fm <- (action tank-pressure high ?val)
     ?ps <- (pump-speed ?wps)
=>
;    (printout t "Entering ACTION-SLOW-MOTOR" crlf)

     (retract ?fm)
     (retract ?ps)

     (if (< ?val (/ ?wps 3000)) then (assert (pump-speed =
                                   (- ?wps (* 100 ?val))))
               else (assert (pump-speed 0)))
```

```
)

;-------------------------------------------------------------------
;                    ACTION-FAST-MOTOR
;
;    This rule will speed up the motor by some variable amount
;    based on the percent belief.
;
;-------------------------------------------------------------------

(defrule action-fast-motor
     (state calc)
     ?fm <- (action tank-pressure low ?val)
     ?ps <- (pump-speed ?wps)
=>
;    (printout t "Entering ACTION-FAST-MOTOR" crlf)
     (retract ?fm)
     (retract ?ps)
     (if (> ?val (/ ?wps 3000)) then (assert (pump-speed =
                                    (+ ?wps (* 100 ?val))))
               else (assert (pump-speed 3000)))
)

;-------------------------------------------------------------------
;                              ACTION-SLOW-MDOT
;
;    This rule will slow down the mass flow rate by some variable amount
;    based on the percent belief.
;
;-------------------------------------------------------------------

(defrule action-slow-mdot
        (state calc)
        ?fm <- (action cabin-pressure high ?val)
        ?ps <- (mass-in ?mi)
=>
;       (printout t "Entering ACTION-SLOW-MDOT" crlf)

        (retract ?fm)
        (retract ?ps)

        (if (< ?val (/ ?mi .5)) then (assert (mass-in =
                                    (- ?mi (* .05 ?val))))
              else (assert (mass-in 0)))
)

;-------------------------------------------------------------------
;                              ACTION-FAST-MDOT
;    This rule will speed up the mass flow rate by some variable amount
;    based on the percent belief.
;
;-------------------------------------------------------------------
```

```
(defrule action-fast-mdot
        (state calc)
        ?fm <- (action cabin-pressure low ?val)
        ?ps <- (mass-in ?mi)
=>
;       (printout t "Entering ACTION-FAST-MDOT" crlf)
        (retract ?fm)
        (retract ?ps)
        (if (> ?val (/ ?mi .5)) then (assert (mass-in =
                                    (+ ?mi (* .05 ?val))))
              else (assert (mass-in .5)))
)

;-------------------------------------------------------------------
;                    CALC-DONE
;
;    This will end the calculations and adjustments made for
;    the fuzzy logic sets.  Notice that there is a salience of -10.
;    This is so that the calculations can complete recursively until
;    they are all done.
;
;-------------------------------------------------------------------

(defrule calc-done
     (declare (salience -10))
     ?s <- (state calc)
=>
;    (printout t "Entering CALC-DONE" crlf)
     (retract ?s)
     (assert (state bosch))
)

;-------------------------------------------------------------------
;                    BOSCH-MASS-FLOW
;
;    This rule will change the mass in the tank used depending
;    on whether the tank pressure is above or below 101.125.
;
;-------------------------------------------------------------------

(defrule bosch-mass-flow

     ?s <- (state bosch)
     (tank-pressure ?tp)
     ?f <- (mtankused ?mt)

=>
;    (printout t "Entering BOSCH-MASS-FLOW" crlf)
     (retract ?s)
     (retract ?f)
     (if (>= ?tp 101.125) then
          (assert (mtankused .000046333))
```

```
      )
      (if (< ?tp 101.125) then
           (assert (mtankused 0))
      )
      (assert (state bosch-sys))
)

;-------------------------------------------------------------------------
;                                  BOSCH-SYSTEM
;
;
;-------------------------------------------------------------------------

(defrule bosch-system

        ?s <- (state bosch-sys)
        (tank-pressure ?tp)
     ?g <- (sys ?sy)
     ?h <- (sys-time ?st)
     ?i <- (mass-in ?mi)

=>
;    (printout t "Entering BOSCH-SYSTEM" crlf)

     (retract ?s)
        (if (and (< ?tp 137.000) (= ?sy 0)) then
          (retract ?g)
          (retract ?h)
          (retract ?i)
          (assert (sys 1))
          (assert (sys-time 0))
          (assert (mass-in .5))
        )
     (assert (state writeedata))
)

;-------------------------------------------------------------------------
;                     WRITE-edata
;
;    Write in the values for the Fortran and simulation modules.
;
;-------------------------------------------------------------------------

(defrule write-edata
     ?s  <- (state writeedata)
     ?ts <- (time-step ?wts)
     ?mi <- (mass-in ?wmi)
     ?pc <- (partial-co2 ?wpc)
     ?tp <- (tank-pressure ?wtp)
     ?ps <- (pump-speed ?wps)
     ?sy <- (sys ?wsy)
     ?st <- (sys-time ?wst)
```

```
        ?st2 <- (sys-2time ?wst2)
        ?co <- (co2-rem ?wco)
        ?pb <- (pb ?wpb)
        ?p2 <- (pbco2 ?wp2)
        ?ma <- (mco2p ?wma)
        ?mc <- (mco2r ?wmc)
        ?va <- (valve ?wva)
        ?mt <- (mtankused ?wmt)

   =>
   ;    (printout t "Entering write expert data" crlf)

        (retract ?s)
        (retract ?ts)
        (retract ?mi)
        (retract ?pc)
        (retract ?tp)
        (retract ?ps)
        (retract ?sy)
        (retract ?st)
        (retract ?st2)
        (retract ?co)
        (retract ?pb)
        (retract ?p2)
        (retract ?ma)
        (retract ?mc)
        (retract ?va)
        (retract ?mt)

        (system "rm edata")
        (open "edata" data "w")
             (printout data ?wts crlf)
             (printout data ?wmi crlf)
             (printout data ?wpc crlf)
             (printout data ?wtp crlf)
             (printout data ?wps crlf)
             (printout data ?wsy crlf)
             (printout data ?wst crlf)
             (printout data ?wst2 crlf)
             (printout data ?wco crlf)
             (printout data ?wpb crlf)
             (printout data ?wp2 crlf)
             (printout data ?wma crlf)
             (printout data ?wmc crlf)
             (printout data ?wva crlf)
             (printout data ?wmt crlf)

      (close data)
)
```

5.7 Simulation Source Code

```
C$NOEXT
C$NOWARN
C$TIME=36000
C$INCLUDE SIM
C$INCLUDE PID
C$INCLUDE EXP
*****
*     CREATED BY:  STAN KROEKER
*     CREATED:     APRIL 1, 1992
*
*     THIS PROGRAM IS TO TEST THE SIMULATION AND THE CONTROLS
*
*****
      PROGRAM COOL
        DOUBLE PRECISION MDOT,MAIR,MCO2R,MCO2P,MTANKUSED,CO2REM,MCO2
        INTEGER SIMTIME,DT,TIME,BLOSPEED,BEDSWITCH,SYSTIME,SYS,VALVE,I
        INTEGER SYSTIME2

C ... REAL DESSICANT BED VALUES

      REAL MDBADS(2,2),MDBTANK(2)

C ... REAL SORBENT BED AND SORBENT PUMP VALUES

      REAL MSBTANK(3),MSBADS(2,3),TBED(2,3)

C ... REAL CO2 TANK VALUES

      DOUBLE PRECISION MTANK,MTIN,MTOUT

*     INITIALIZE VARIABLES
        SIMTIME = 0
        SYSTIME2 = 0
        DT = 6
        MDOT = 0
        PRESS = 101.125
        TEMP = 300
        PHI = .78
        TIME = 0
        PCO2 = .0667
        BLOSPEED = 0
        BEDSWITCH = 0
        PTANK = 300
        PUMPSPEED = 0
        PTANK_SET = 350
        PTANK_OLD = 101.125
        SYS = 0
        SYSTIME = 0
        CO2REM = 0
        PB = 101.125
```

```
        PBCO2 = .0667
        MAIR = 0
        MCO2R = 0
        MCO2P = 0
        VALVE = 0
        MTANKUSED = 0
        I=0
        J=0
      CPAIR=1.006
      MDBTANK(1)=100.
      MDBTANK(2)=100.
      MDBADS(1,1)=0.001
      MDBADS(1,2)=0.001
      MDBADS(2,1)=0.001
      MDBADS(2,2)=0.001

C ... BLOWER/PRECOOLER SUBROUTINE VARIABLE INITIALIZATION

10    CPH2O=4.184
      EPSILON=.8
      PAIR=1.1614
      TWATER=288
      VELAIR=3.

C ... SORBENT BED AND SORBENT PUMP SUBROUTINE VARIABLE
C ... INITIALIZATION

      CPAIR=1.006
      CVAIR=.719
      CVBED=1.
      CVCO2=.7
      HCO2=572.
      HEAT=0.
      MCO2=0.
      MAIR=.1
      MSBADS(1,1)=0.
      MSBADS(1,2)=0.
      MSBADS(1,3)=0.
      MSBADS(2,1)=0.
      MSBADS(2,2)=0.
      MSBADS(2,3)=0.
      MSBTANK(1) = 30.
      MSBTANK(2) = 30.
      MSBTANK(3) = 30.
      PBED=101.325
      PSPIN=101.325
      RAIR=.287
      RCO2=.1889
      SPEED=3000
      TBED(1,1)=300.
      TBED(1,2)=300.
      TBED(1,3)=300.
```

```
      TBED(2,1)=300.
      TBED(2,2)=300.
      TBED(2,3)=300.
      VBED=.1
      VPUMP = 0.0006

C ... CO2 TANK SUBROUTINE VARIABLE INITIALIZATION

      MTANK=2.
      TTANK=300.
      VTANK=1.
      MTOUT=0.

      WRITE(*,*)'Working...'

      DO 100  TIME = 0,7500000,DT

         IF(MOD(TIME,3600).EQ.0) THEN
             WRITE(*,*)'The time now is ',J,' hours....SYS = ',SYS
             J=J+1

             IF(J.EQ.25) STOP

         ENDIF
C
C DETERMINE CONTROLLER TO UTILIZE
C
         CALL PID(SIMTIME,DT,MDOT,PRESS,TEMP,PHI,TIME,PCO2,
     &            BLOSPEED,SYSTIME2,PTANK,PUMPSPEED,PTANK_SET,
     &            PTANK_OLD,SYS,SYSTIME,CO2REM,PB,PBCO2,MCO2R,
     &            MCO2P,VALVE,MTANKUSED)

         CALL EXP(SIMTIME,DT,MDOT,PRESS,TEMP,PHI,TIME,PCO2,
     &            BLOSPEED,SYSTIME2,PTANK,PUMPSPEED,PTANK_SET,
     &            PTANK_OLD,SYS,SYSTIME,CO2REM,PB,PBCO2,MCO2R,
     &            MCO2P,VALVE,MTANKUSED)

C
C CALL MAIN SIMULATION
C
         CALL SIM(SIMTIME,DT,MDOT,PRESS,TEMP,PHI,TIME,
     &               PCO2,BLOSPEED,MCO2,MAIR,PTANK,PUMPSPEED,
     &               PTANK_SET,PTANK_OLD,SYS,SYSTIME2,CO2REM,
     &               PB,PBCO2,MCO2R,MCO2P,VALVE,MTANKUSED,CPAIR,
     &               MDBTANK,MDBADS,CPH2O,EPSILON,PAIR,TWATER,VELAIR,
     &               CVAIR,CVBED,CVCO2,HCO2,HEAT,MSBADS,MSBTANK,
     &               RAIR,RCO2,TBED,VBED,VPUMP,MTANK,TTANK,VTANK,ADSORB,
     &               DESORB,CO2ADSORB,CO2DESORB)
       I=I+1

100    CONTINUE
       END
```

```
C .....................................................................
C ... PROGRAM NAME:  SIM.FOR
C ... GROUP NAME:  CLASSICAL CONTROL
C ... CREATED BY:  PAUL M. SNIDER, STAN KROEKER
C ..................  MARCH 3, 1992
C ... REVISED:  03/10/92 (PS)
C ... SUBROUTINE CALLS:  DESSBED.FOR
C ......................  BLOWCOOL.FOR
C ......................  SORBED.FOR
C ......................  SORPUMP.FOR
C ......................  CO2TANK.FOR
C ......................  CWRITE.FOR -- (IN EXPERT SIM ONLY)
C ... CALLED FROM:  CO2REM.FOR
C ... INPUT FILE:  CREAD.FOR
C ... OUTPUT FILE:  CWRITE.FOR
C ........................  LIST OF VARIABLES
C .....................................................................
C ... SEE SUBROUTINES FOR LOCAL VARIABLE LIST
C .....................................................................
C$NOEXT
C$NOWARN
C$INCLUDE DESSBED
C$INCLUDE BLOWCOOL
C$INCLUDE CWRITE
C$INCLUDE SORBED
C$INCLUDE SORPUMP
C$INCLUDE CO2TANK
C$INCLUDE CABIN
C$TIME=3600

      SUBROUTINE SIM(SIMTIME,DT,MDBIN,PDBIN,TDBIN,PHIIN,SYSTIME,
     &               PCO2,BLOSPEED,MCO2,MAIR,PTANK,SPEED,
     &               PTANK_SET,PTANK_OLD,SYS,TIME,CO2REM,
     &               PBED,PVCO2,MCO2R,MCO2P,VALVE,MTOUT,CPAIR,
     &               MDBTANK,MDBADS,CPH2O,EPSILON,PAIR,TWATER,VELAIR,
     &               CVAIR,CVBED,CVCO2,HCO2,HEAT,MSBADS,MSBTANK,
     &               RAIR,RCO2,TBED,VBED,VPUMP,MTANK,TTANK,VTANK,ADSORB,
     &               DESORB,CO2ADSORB,CO2DESORB)

C ... REAL DESSICANT BED VALUES

      REAL MDBADS(2,2),MDBTANK(2)
      DOUBLE PRECISION MH2O,MH2OR,MDBIN,MDBOUT1,MDBOUT2,MDBOUT3

C ... REAL BLOWER/PRECOOLER VALUES

      DOUBLE PRECISION MBCIN,MBCOUT

C ... REAL SORBENT BED AND SORBENT PUMP VALUES

      REAL MSBTANK(3),MSBADS(2,3),TBED(2,3)
      DOUBLE PRECISION MAIR,MCO2,MCO2R,MCO2P,MSBOUT,MSPOUT,CO2REM
```

```
C ... REAL CO2 TANK VALUES

      DOUBLE PRECISION MTANK,MTIN,MTOUT

C ... DECLARE ALL INTEGERS FOR THE SIMULATION

      INTEGER DT,T,TIME,BLOSPEED,BEDSWITCH,SIMTIME,SYS,SYSTIME,VALVE

C ... RUN CO2 REMOVAL ASSEMBLY SIMULATION AT A TIME STEP DT FOR A
C ... TIME DURATION T
         CALL CABIN(TDBIN,PDBIN,PHIIN,DT,SYSTIME,CO2REM,PCO2)
       IF(SYS.EQ.1) THEN
         CALL DESSBED(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT1,PDBOUT1,PHI
     &                 OUT1,TDBOUT1,CPAIR,DT,MDBADS,MH2O,MH2OR,MDB
     &                 TANK,1,TIME,ADSORB)
         CALL BLOWCOOL(MDBOUT1,PDBOUT1,TDBOUT1,MBCOUT,PBCOUT,TBCO
     &                  UT,CPH2O,EPSILON,PAIR,TWATER,VELAIR)
         CALL SORBED(MBCOUT,PBCOUT,TBCOUT,MSBOUT,PSBOUT,TSBOUT,CP
     &               AIR,CVBED,DT,HCO2,MSBADS,MSBTANK,PCO2
     &               ,PVCO2OUT,TBED,TIME,CO2ADSORB,MCO2R)
         CALL DESSBED(MBCOUT,PBCOUT,PHIOUT1,TBCOUT,MDBOUT3,PDBOUT3,PHI
     &                 OUT3,TDBOUT3,CPAIR,DT,MSBADS,MH2O,MH2OR,MDB
     &                 TANK,3,TIME,ADSORB2)
         CALL SORPUMP(PBCOUT,MSPOUT,PSPOUT,TSPOUT,CVAIR,CVBE
     &                 D,CVCO2,DT,HCO2,HEAT,MCO2,MSBADS,MAIR,MSBTA
     &                 NK,PBED,PVCO2,RAIR,RCO2,SPEED,TBED,TIME,VBE
     &                 D,VPUMP,CO2DESORB,MCO2P)
           IF(VALVE.EQ.1) THEN
             MSBOUT = MSBOUT
             MSPOUT = 0
           ENDIF

         CALL DESSBED(MSBOUT,PSBOUT,PHIOUT1,TSBOUT,MDBOUT2,PDBOUT
     &                 2,PHIOUT2,TDBOUT2,CPAIR,DT,MDBADS,MH2O,MH2O
     &                 R,MDBTANK,2,TIME,DESORB)
        ELSE
          HEAT=0
          MCO2R=0
          MCO2P=0
          MSPOUT=0
          PSPOUT=0
          TSPOUT=0
        ENDIF
          CALL CO2TANK(MSPOUT,PSPOUT,TSPOUT,MTOUT,PTOUT,TTOUT,MTAN
     &                 K,PTANK,TTANK,CVCO2,DT,RCO2,TIME,VTANK)

        RETURN
        END
```

```
C ..............................................................
C ... PROGRAM NAME:  DESSBED.FOR
C ... CALLED FROM:  MAIN PROGRAM  CO2REM.FOR
C ... GROUP NAME:  CLASSICAL CONTROL
C ... CREATED BY:  TIM SPRECKER
C ................. DAN WALDECK
C ................. PAUL M. SNIDER
C ................. FEBRUARY 18, 1992
C ... REVISED:
C ... CALLS:  SUBROUTINE G1Z2
C ... CALLED FROM:  SUBROUTINE DESSBED.FOR
C ... OUTPUT FILE:  NONE
C ..................... LIST OF VARIABLES
C ..............................................................
C ... CPAIR: SPECIFIC HEAT OF AIR (kJ/kg*K)
C ... DT: DELTA TIME STEP (s)
C ... HDBIN: ENTHALPY OF CONDENSATION (kJ/kg)
C ... DBLOAD: LOAD ON THE DESSICANT BEDS
C ... MDBADS(2,2): MASS OF WATER ADSORBED BY DESSICANT (kg)
C ... MDBIN: MASS OF CABIN AIR INTO THE DESSICANT BEDS (kg/s)
C ... MDBOUT: MASS OF CABIN AIR OUT OF THE DESSICANT BEDS (kg/s)
C ... MH2O: MASS OF WATER IN THE INCOMING CABIN AIR (kg)
C ... MH2OR: MASS OF WATER REMOVED BY THE DESSICANT (kg)
C ... MDBTANK(2): MASS OF DESSICANT MATERIAL (kg)
C ... M: INTEGER BED SWITCHING VARIABLE
C ... N: INTEGER BED SWITCHING VARIABLE
C ... PDBIN: PRESSURE OF CABIN AIR INTO THE DESSICANT BEDS (kPa)
C ... PDBOUT: PRESSURE OF CABIN AIR OUT OF THE DESSICANT BEDS
C ............ TO THE BLOWER/PRECOOLER (kPa)
C ... PHIIN: RELATIVE HUMIDITY OF CABIN AIR INTO THE DESSICANT
C ........... BEDS (%/100)
C ... PHIOUT: RELATIVE HUMIDITY OF CABIN AIR OUT OF THE
C ............ DESSICANT BEDS TO THE BLOWER/PRECOOLER (%/100)
C ... PVH2O: PARTIAL PRESSURE OF WATER VAPOR (kPa)
C ... T1: TEMPORARY TEMPERATURE VARIABLE FOR ITERATION (deg K)
C ... T2: TEMPORATY TEMPERATURE VARIABLE FOR ITERATION (deg K)
C ... TDBIN: TEMPEATUTE OF CABIN AIR INTO THE DESSICANT BEDS (K)
C ... TDBOUT: TEMPERATURE OF CABIN AIR OUT OF THE DESSICANT BEDS
C ............ TO THE BLOWER/PRECOOLER (K)
C ... TIME: PRESENT SIMULATION TIME (s)
C ... W: ABSOLUTE HUMIDITY OF CABIN AIR
C..............................................................

C$NOEXT
C$NOWARN

      SUBROUTINE DESSBED(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT,PDBOUT,PH
     &                   IOUT,TDBOUT,CPAIR,DT,MDBADS,MH2O,MH2OR,M
     &                   DBTANK,N,TIME,DBLOAD)

C ... INITIALIZE PARAMETERS
```

```
      REAL MDBADS(2,2),MDBTANK(2)
      DOUBLE PRECISION MH2O,MH2OR,MDBIN,MDBOUT
      INTEGER DT,TIME
      M=MOD(TIME/1800,2)
      MDBIN=MDBIN*DT

      IF(N.EQ.1)THEN

C ... CALCULATE PARTIAL PRESSURE OF WATER VAPOR

        PVH2O=PHIIN*PSAT(TDBIN-273)

C ... CALCULATE OMEGA IN KG VAPOR/KG DRY AIR

        W=(.622*PVH2O)/(PDBIN-PVH2O)

C ... BREAKDOWN AIR COMPONENTS

        MH2O=W*MDBIN/(1+W)

C ... CALL ZEOLITE SUBROUTINE FOR ADSORPTION AND DESORPTION
        IWEIGHT = 1
        CALL G1Z2(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT,PDBOUT,PHIOUT,TD
     &            BOUT,CPAIR,DT,MDBADS(M+1,1),MH2O,MH2OR,MDBTANK(
     &       .    1),1,TIME,DBLOAD,IWEIGHT)
        CALL G1Z2(MDBOUT,PDBOUT,PHIOUT,TDBOUT,MDBOUT,PDBOUT,PHIOUT,TD
     &            BOUT,CPAIR,DT,MDBADS(M+1,2),MH2O,MH2OR,MDBTANK(
     &            2),2,TIME,DBLOAD,IWEIGHT)

      ELSE IF(N.EQ.2) THEN
        IWEIGHT = 1
        TDBIN=363
        CALL G1Z2(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT,PDBOUT,PHIOUT,TD
     &            BOUT,CPAIR,DT,MDBADS(2-M,2),MH2O,MH2OR,MDBTANK(
     &            2),2,TIME,DBLOAD,IWEIGHT)
        CALL G1Z2(MDBOUT,PDBOUT,PHIOUT,TDBOUT,MDBOUT,PDBOUT,PHIOUT,TD
     &            BOUT,CPAIR,DT,MDBADS(2-M,1),MH2O,MH2OR,MDBTANK(
     &            1),1,TIME,DBLOAD,IWEIGHT)

      ELSE
        IWEIGHT = 0
        CALL G1Z2(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT,PDBOUT,PHIOUT,TD
     &            BOUT,CPAIR,DT,MDBADS(M+1,2),MH2O,MH2OR,MDBTANK(
     &            2),2,TIME,DBLOAD,IWEIGHT)
c        write(*,*)mh2or,phiin,phiout
      ENDIF
      MDBIN=MDBIN/DT
      MDBOUT=MDBOUT/DT

      RETURN
      END
```


C-2

```
      SUBROUTINE G1Z2(MDBIN,PDBIN,PHIIN,TDBIN,MDBOUT,PDBOUT,PHIOU
     &                T,TDBOUT,CPAIR,DT,MDBADS,MH2O,MH2OR,MDBTANK
     &                ,N,TIME,DBLOAD,IWEIGHT)

C ... INITIALIZE PARAMETERS

      REAL MDBADS,MDBTANK
      DOUBLE PRECISION MH2O,MH2OR,MDBIN,MDBOUT
      INTEGER DT,TIME

C ... DETERMINE CURRENT TANK LOAD
      ICOUNT=0
      DBLOAD=MDBADS/MDBTANK

C ... FIGURE AIR CHARACTERISTICS AT EQUILIBRIUM WITH DESICCANT

      IF(N.EQ.1) PHIOUT=DBLOAD/.5263
      IF(N.EQ.2.AND.DBLOAD.LE..17) PHIOUT=.4*DBLOAD
      IF(N.EQ.2.AND.DBLOAD.GT..17) PHIOUT=.068+40*(DBLOAD-.17)
      T1=TDBIN

C ... CALCULATE PARTIAL PRESSURE OF WATER VAPOR
 10   PVH2O=PHIOUT*PSAT(T1-273)
      W=(.622*PVH2O)/(PDBIN-PVH2O)

C ... CALCULATE WATER REMOVED

      IF(IWEIGHT.EQ.1) THEN
         MH2OR=MH2O-W*(MDBIN-MH2O)
       ELSE
         MH2OR=5*(MH2O-W*(MDBIN-MH2O))
      ENDIF

C ... ADD ENERGY OF EVAPORATION TO AIR AND FIND NEW TEMPERATURE
      HDBIN=MH2OR*(2502-2.389*(TDBIN-273))
      T2=TDBIN+HDBIN/(MDBIN-MH2OR)/CPAIR

C ... ITERATE TO FIND EQUILIBRIUM TEMPERATURE
        IF(ABS(T2-T1).GT.1)THEN
C ..... WEIGHT THE AVERAGE TOWARD T1 FOR STABILITY
        T1=(3*T1+T2)/4
        ICOUNT=ICOUNT+1
C ... QUIT WITH CURRENT T1 AFTER 50TH TRY
        IF(ICOUNT.LT.50)GOTO 10
      ENDIF

      TDBOUT=T1
      PDBOUT=PDBIN

C ... ADD REMOVED WATER TO BED
```

```
      MDBADS=MDBADS+MH2OR

C ... FIGURE NEW MAKEUP OF AIR
      MH2O=MH2O-MH2OR
      MDBOUT=MDBIN-MH2OR

      RETURN
      END

C ... WATER VAPOR SATURATION PRESSURE (kPa) AT TEMPERATURE (deg C)
      FUNCTION PSAT(T)
      PSAT=.3972+.0629*T+.001099*T**2+.00001705*T**3+.0000006192*T**4
      END
```

```
C .....................................................................
C ... PROGRAM NAME:  BLOWCOOL.FOR
C ... CALLED FROM:  MAIN PROGRAM  CO2REM.FOR
C ... GROUP NAME:  CLASSICAL CONTROL
C ... CREATED BY:  CARL ALBRECHT
C ..................  ROGER BURJES
C ..................  PAUL M. SNIDER
C ..................  MARCH 12, 1992
C ... REVISED:
C ... SUBROUTINE CALLS:  NONE
C ... OUTPUT FILE:  NONE
C ........................  LIST OF VARIABLES
C .....................................................................
C ... CPH2O: SPECIFIC HEAT OF AIR (kJ/kg*K)
C ... EPSILON: RADIATIVE HEAT TRANSFER COEFFICIENT (unitless)
C ... MBCIN: MASS OF DESSICANT BED AIR INTO THE BLOWER (kg)
C ... MBCOUT: MASS OF AIR OUT OF THE PRECOOLER (kg/s)
C ... PAIR: DENSITY OF CABIN AIR (kg/m^3)
C ... PBCIN: PRESSURE OF DESSICANT BED AIR INTO THE BLOWER (kPa)
C ... PBCOUT: PRESSURE OF AIR OUT OF THE PRECOOLER (kPa)
C ... QDOT: HEAT TRANSFER BETWEEN BLOWER AIR AND PRECOOLER (kJ/s)
C ... TBCIN: TEMPERATURE OF DESSICANT BED AIR INTO THE BLOWER (K)
C ... TBCOUT: TEMPERATURE OF AIR OUT OF THE PRECOOLER (K)
C ... TWATER: TEMPERATURE OF COOLANT (WATER) IN THE PRECOOLER (K)
C ... VELAIR: VELOCITY OF CABIN AIR THROUGH THE BLOWER (m/s)
C .....................................................................

C$NOEXT
C$NOWARN

      SUBROUTINE BLOWCOOL(MBCIN,PBCIN,TBCIN,MBCOUT,PBCOUT,TBCOUT,
     &                    CPH2O,EPSILON,PAIR,TWATER,VELAIR)

      DOUBLE PRECISION MBCIN,MBCOUT

C ... CALCULATE HEAT TRANSFER BETWEEN BLOWER AIR AND COOLANT

      QDOT=EPSILON*CPH2O*MBCIN*(TBCIN-TWATER)
      MBCOUT=MBCIN

      PBCOUT = PBCIN

C ... FIND NEW BLOWER AIR TEMPERATURE

      TBCOUT=TBCIN-QDOT/(CPH2O*MBCOUT)

      RETURN
      END
```

```
C$NOEXT
C$NOWARN
C$INCLUDE FUNCZ5A.FOR


      SUBROUTINE SORBED(MSBIN,PSBIN,TSBIN,MSBOUT,PSBOUT,TSBOUT,CP
     &                  AIR,CVBED,DT,HCO2,MSBADS,MSBTANK,PVC
     &                  O2IN,PVCO2OUT,TBED,TIME,SBLOAD,MCO2R)

      REAL MSBTANK(3),MSBADS(2,3),TBED(2,3)
      DOUBLE PRECISION MCO2,MCO2R,MSBIN,MSBOUT
      INTEGER DT,TIME
      M=MOD(TIME/1800,2)
      MSBIN=MSBIN*DT
      MCO2=1.519*PVCO2IN/PSBIN*MSBIN

C ... DETERMINE CURRENT TANK LOAD

      SBLOAD=100*MSBADS(M+1,3)/MSBTANK(3)

C ... FIGURE AIR CHARACTERISTICS AT EQUILIBRIUM WITH SORBENT
      PVCO2OUT=Z5A(SBLOAD,TBED(M+1,3)-273)
      W=1.519*PVCO2OUT/(PSBIN-PVCO2OUT)

C ... CALCULATE CO2 REMOVED

      MCO2R=MCO2-W*(MSBIN-MCO2)
      IF(-MCO2R.GT.MSBADS(M+1,3)) MCO2R=-MSBADS(M+1,3)
C ... ADD REMOVED CO2 TO BED

      MSBADS(M+1,3)=MSBADS(M+1,3)+MCO2R

C ... FIGURE NEW MAKEUP OF AIR

      MSBOUT=MSBIN-MCO2R

C ... ADD ENERGY OF EVAPORATION TO AIR+BED AND FIND NEW
C ... TEMPERATURE

      TSBOUT=(MCO2R*HCO2+MSBOUT*CPAIR*TSBIN+(MSBTANK(3)+MSBA
     &       DS(M+1,3))*CVBED*TBED(M+1,3))/(MSBOUT*CPAIR+(MSBTANK
     &       (3)+MSBADS(M+1,3))*CVBED)
      TBED(M+1,3)=TSBOUT
      PSBOUT=PSBIN
      MSBIN=MSBIN/DT
      MSBOUT=MSBOUT/DT

      RETURN
      END
```

```
C$NOEXT
C$NOWARN

      SUBROUTINE SORPUMP(PSPIN,MSPOUT,PSPOUT,TSPOUT,CVAIR,
     &                   CVBED,CVCO2,DT,HCO2,HEAT,MCO2,MSBADS,MAIR,
     &                   MSBTANK,PBED,PVCO2,RAIR,RCO2,SPEED,TBED,
     &                   TIME,VBED,VPUMP,SBLOAD,MCO2P)

      REAL MSBTANK(3),MSBADS(2,3),TBED(2,3)
      DOUBLE PRECISION MSPOUT,MCO2P,MAIR,MCO2,MCO2R
      INTEGER DT,TIME
      M=MOD(TIME/1800,2)
C ... DETERMINE CURRENT TANK LOAD
      SBLOAD=100*MSBADS(2-M,3)/MSBTANK(3)
      IF(M.NE.MOD((TIME-DT)/1800,2)) THEN
        PBED=PSPIN
        HEAT=12.*DT

C ... FIGURE INITIAL AIR CHARACTERISTICS AT EQUILIBRIUM WITH
C ... SORBENT

        PVCO2=Z5A(SBLOAD,TBED(2-M,3)-273)
        MCO2=PVCO2*VBED/RCO2/TBED(2-M,3)
        MAIR=(PBED-PVCO2)*VBED/RAIR/TBED(2-M,3)
      ENDIF

      IF(TBED(2-M,3).GT.478.AND.HEAT.GT.0.) HEAT=0.*DT
      IF(TBED(2-M,3).GT.368.AND.MOD(TIME,1800).GE.1480) HEAT=-12.*
     &   DT


C ... ADD ENERGY INPUT TO AIR+BED AND FIND NEW TEMPERATURE AND
C ... PVCO2

      TBED(2-M,3)=TBED(2-M,3)+HEAT/(MAIR*CVAIR+MCO2*CVCO2+(MSBTAN
     &            K(3)+MSBADS(2-M,3))*CVBED)
      PVCO2=Z5A(SBLOAD,TBED(2-M,3)-273)

C ... CALCULATE CO2 REMOVED FROM BED

      MCO2R=PVCO2*VBED/RCO2/TBED(2-M,3)-MCO2

C ... SUBTRACT REMOVED CO2 FROM BED

      MSBADS(2-M,3)=MSBADS(2-M,3)-MCO2R
      MCO2=MCO2+MCO2R

C ... ADD ENERGY OF EVAPORATION TO AIR+BED AND FIND NEW
C ... TEMPERATURE

      TBED(2-M,3)=TBED(2-M,3)-MCO2R*HCO2/(MAIR*CVAIR+MCO2*CVCO2+(
     &            MSBTANK(3)+MSBADS(2-M,3))*CVBED)
```

```
C ... RUN CO2 PUMP

      MCO2P=MCO2*(VPUMP*SPEED*DT/60/(VBED+VPUMP*SPEED*DT/60.))
      MSPOUT=(MCO2+MAIR)*(VPUMP*SPEED*DT/60/(VBED+VPUMP*SPEED*
     &  DT*60.))
      PSPOUT=PVCO2+MAIR*RAIR*TBED(2-M,3)/VBED
      TSPOUT=TBED(2-M,3)

      MCO2=MCO2*(1-VPUMP*SPEED*DT/60/(VBED+VPUMP*SPEED*DT/60.))

      MAIR=MAIR*(1-VPUMP*SPEED*DT/60/(VBED+VPUMP*SPEED*DT/60.))


C ... DETERMINE CURRENT TANK LOAD

      SBLOAD=100*MSBADS(2-M,3)/MSBTANK(3)

C ... FIGURE AIR CHARACTERISTICS AT EQUILIBRIUM WITH SORBENT

      PVCO2=Z5A(SBLOAD,TBED(2-M,3)-273)

C ... CALCULATE CO2 REMOVED FROM BED

      MCO2R=PVCO2*VBED/RCO2/TBED(2-M,3)-MCO2

C ... SUBTRACT REMOVED CO2 FROM BED

      MSBADS(2-M,3)=MSBADS(2-M,3)-MCO2R
      MCO2=MCO2+MCO2R

C ... ADD ENERGY OF EVAPORATION TO AIR+BED AND FIND NEW
C ... TEMPERATURE

      TBED(2-M,3)=TBED(2-M,3)-MCO2R*HCO2/(MAIR*CVAIR+MCO2*CVCO2+(
     &            MSBTANK(3)+MSBADS(2-M,3))*CVBED)

      PBED=PVCO2+MAIR*RAIR*TBED(2-M,3)/VBED

      RETURN
      END
```

```
C ..............................................................
C ... PROGRAM NAME:  CO2TANK.FOR
C ... CALLED FROM:  MAIN PROGRAM  CO2REM.FOR
C ... GROUP NAME:  CLASSICAL CONTROLS
C ... CREATED BY:  DANIEL T. WALDECK
C ..................PAUL M. SNIDER
C ..................MARCH 29, 1992
C ... REVISED:
C ... CALLS:  NONE
C ... OUTPUT FILE:  NONE
C ..............................................................
C$NOEXT
C$NOWARN

      SUBROUTINE CO2TANK(MTIN,PTIN,TTIN,MTOUT,PTOUT,TTOUT,MTANK,P
     &                   TANK,TTANK,CVCO2,DT,RCO2,TIME,VTANK)

      DOUBLE PRECISION MTIN,MTOUT,MTANK
      INTEGER DT,TIME

C ... CALCULATE CURRENT TANK CONDITIONS

      TTANK=TTANK+MTIN*CVCO2*TTIN/(MTIN+MTANK)
      PTANK=MTANK*RCO2*TTANK/VTANK

C ... CALCULATE NEW MASS OF THE TANK

      MTANK=MTANK+MTIN-MTOUT

C ... CALCULATE TANK OUTPUT CONDITIONS

      TTOUT=TTANK-MTOUT*CVCO2*TTANK/(MTANK)
      PTOUT=PTANK

      RETURN
      END
```

5.8 Graphical Interface Code

```
/*
 *Filename:show.h
 *
 *   Creator:  Carl Albrect
 *
 *   Created:  March 31, 1992
 *   Modified: May 9, 1992    - Robert Swenson - added bed_switch
 *
 *   Description:
 *             types and constants for the show program
 */


struct diag_data {
     int CO2,
         Tank_Pres,
         Pump_Speed,
         Absorb,
            Desorb,
            CO2_Ab,
         CO2_De,
         sys,
         fan;
     int oldCO2,          /* what the graph was last time step */
         oldTank_Pres,
         oldPump_Speed,
         oldAbsorb,
         oldDesorb,
         oldCO2_Ab,
         oldCO2_De;
     int bed_switch,           /* 0 if top bed = desorb, 1 otherwise */
         old_bed;
     };
```

```
/*\
 *   Filename: show.c
 *
 *   Created:  March 15, 1992 - converted from bars.c
 *   Modified: May 9, 1992    Robert Swenson - added code.bed_switch
 *
 *   creator:  Robert Swenson
 *
 *   Description:
 *        create a window, list NASA/USRA ADT members, show overview of
 *        carbon dioxide removal system, and show pressure bars/graphs
\*/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include "show.h"
#include <stdio.h>

#include "icon_bitmap"

#define BITMAPDEPTH 1
#define TOO_SMALL 0
#define BIG_ENOUGH 1

Display *display;
int screen_num;

static char *progname; /* name this program was invoked by (argv[0]) */

void main(argc, argv)
int argc;
char **argv;
{
     Window win;
     unsigned int width, height;        /* window size */
     int x, y;                       /* window position */
     unsigned int border_width = 4;          /* four pixels */
     unsigned int display_width, display_height;
     unsigned int icon_width, icon_height;
     char *window_name = "Simulation Demonstration";
     char *icon_name = "Simulation Demo";
     Pixmap icon_pixmap;
     XSizeHints size_hints;
     XIconSize *size_list;
     int count;
     FILE *my_data;
     XEvent report;
     GC gc;
```

```
XFontStruct *font_info;
char *display_name = NULL;
int window_size = BIG_ENOUGH; /* or TOO_SMALL to display contents */
struct diag_data code;

progname = argv[0];

/* connect to X server */
if ( (display=XOpenDisplay(display_name)) == NULL )
{
     (void) fprintf( stderr, "%s: cannot connect to X server %s\n",
                progname, XDisplayName(display_name));
     exit( -1 );
}

/* get screen size from display structure macro */
screen_num = DefaultScreen(display);
display_width = DisplayWidth(display, screen_num);
display_height = DisplayHeight(display, screen_num);

/* Note that in a real application, x and y would default to 0
 * but would be settable from the command line or resource database.
 */
x = y = 0;

/* size window with enough room for text */
width = 640, height = 460;

/* create opaque window */
win = XCreateSimpleWindow(display, RootWindow(display,screen_num),
          x, y, width, height, border_width, BlackPixel(display,
          screen_num), WhitePixel(display,screen_num));

/* Get available icon sizes from Window manager */

if (XGetIconSizes(display, RootWindow(display,screen_num),
          &size_list, &count) == 0)
     (void) fprintf( stderr,
     "%s: Window manager didn't set icon sizes - using default.\n",
          progname);
else { ; /* A real application would search through size_list
      * here to find an acceptable icon size, and then
      * create a pixmap of that size.  This requires
      * that the application have data for several sizes
      * of icons. */
}

/* Create pixmap of depth 1 (bitmap) for icon */
icon_pixmap = XCreateBitmapFromData(display, win, icon_bitmap_bits,
          icon_bitmap_width, icon_bitmap_height);
```

```
size_hints.flags = PPosition | PSize | PMinSize;
size_hints.min_width = width;
size_hints.min_height = height;
{
XWMHints wm_hints;
XClassHint class_hints;

/* format of the window name and icon name args has changed in R4 */
XTextProperty windowName, iconName;

if (XStringListToTextProperty(&window_name, 1, &windowName) == 0) {
     (void) fprintf( stderr,
     "%s: structure allocation for windowName failed.\n", progname);
     exit(-1);
}

if (XStringListToTextProperty(&icon_name, 1, &iconName) == 0) {
     (void) fprintf( stderr,
     "%s: structure allocation for iconName failed.\n", progname);
     exit(-1);
}

wm_hints.initial_state = NormalState;
wm_hints.input = True;
wm_hints.icon_pixmap = icon_pixmap;
wm_hints.flags = StateHint | IconPixmapHint | InputHint;

class_hints.res_name = progname;
class_hints.res_class = "Simulation Demo";

XSetWMProperties(display, win, &windowName, &iconName,
          argv, argc, &size_hints, &wm_hints,
          &class_hints);
}

/* Select event types wanted */
XSelectInput(display, win, ExposureMask | KeyPressMask |
          ButtonPressMask | StructureNotifyMask);

load_font(&font_info);

/* create GC for text and drawing */
getGC(win, &gc, font_info);

/* Display window */
XMapWindow(display, win);

/* open file */
if ((my_data = fopen("data.sim","r")) == NULL)
{
     fprintf(stderr, "bar: unable to open data.sim\n");
     exit(-1);
```

```
}
     /* init the structure */
code.CO2 = 0;
code.Tank_Pres = 0;
code.Pump_Speed = 0;
code.Absorb = 0;
code.Desorb = 0;
code.CO2_Ab = 0;
code.CO2_De = 0;
code.fan    = 0;
code.bed_switch     = 0;

     /* init the window */
   draw_text(win, gc, font_info, width, height, code.bed_switch);
   draw_graphics(win, gc, width, height, &code);


/* get events, use first to display text and graphics */
/* To run through the data file infinity times */
while (1 == 1)
{
if (feof(my_data))
{
     fseek(my_data,0,0);
}
               /* don't forget the old data */
    code.oldCO2 = code.CO2;
    code.oldTank_Pres = code.Tank_Pres;
    code.oldPump_Speed = code.Pump_Speed;
    code.oldAbsorb = code.Absorb;
    code.oldDesorb = code.Desorb;
    code.oldCO2_Ab = code.CO2_Ab;
    code.oldCO2_De = code.CO2_De;

    if (!XPending(display))   /* if nothing to do then just draw */
    {
               /* get the next piece of data */
     fscanf(my_data, "%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\n", &code.CO2,
          &code.Tank_Pres, &code.Pump_Speed, &code.Absorb,
          &code.Desorb, &code.CO2_Ab, &code.CO2_De, &code.sys);

     code.bed_switch = 0;
     if (code.bed_switch != code.old_bed)
     {
          /* erase the window and re-draw with the new beds */
     }

     draw_data(win,gc,width,height,&code);
    }
    else
    {                    /* else update the win or whatever */
     XNextEvent(display, &report);
```

```
        switch  (report.type) {
        case Expose:
             /* unless this is the last contiguous expose,
              * don't draw the window */
             if (report.xexpose.count != 0)
                  break;

             /* if window too small to use */
             /*if (window_size == TOO_SMALL)
                  TooSmall(win, gc, font_info);
             else*/ {
                  /* place text in window */
                  draw_text(win, gc, font_info, width, height,
                            code.bed_switch);

                  /* place graphics in window, */
                  draw_graphics(win, gc, width, height, &code);
             }
             break;
        case ConfigureNotify:
             /* window has been resized, change width and
              * height to send to draw_text and draw_graphics
              * in next Expose */
             width = report.xconfigure.width;
             height = report.xconfigure.height;
             if ((width < size_hints.min_width) ||
                       (height < size_hints.min_height))
                  window_size = TOO_SMALL;
             else
                  window_size = BIG_ENOUGH;
             break;
        case ButtonPress:
             /* trickle down into KeyPress (no break) */
        case KeyPress:
             XUnloadFont(display, font_info->fid);
             XFreeGC(display, gc);
             XCloseDisplay(display);
             exit(1);
        default:
             /* all events selected by StructureNotifyMask
              * except ConfigureNotify are thrown away here,
              * since nothing is done with them */
             break;
        } /* end switch */
      } /* XPending */
} /* end while */

fclose(my_data);          /* we're done with the data */
{
    char ch;
    ch = getchar();
}
```

```
} /* main */


getGC(win, gc, font_info)
Window win;
GC *gc;
XFontStruct *font_info;
{
     unsigned long valuemask = 0; /* ignore XGCvalues and use defaults */
     XGCValues values;
     unsigned int line_width = 3;
     int line_style = LineSolid;
     int cap_style = CapRound;
     int join_style = JoinRound;
     int dash_offset = 0;
     static char dash_list[] = {12, 24};
     int list_length = 2;

     /* Create default Graphics Context */
     *gc = XCreateGC(display, win, valuemask, &values);

     /* specify font */
     XSetFont(display, *gc, font_info->fid);

     /* specify black foreground since default window background is
      * white and default foreground is undefined. */
     XSetForeground(display, *gc, BlackPixel(display,screen_num));

     /* set line attributes */
     XSetLineAttributes(display, *gc, line_width, line_style,
                cap_style, join_style);
} /* getGC */


load_font(font_info)
XFontStruct **font_info;
{
     char *fontname = "9x15";

     /* Load font and get font information structure. */
     if ((*font_info = XLoadQueryFont(display,fontname)) == NULL)
     {
          (void) fprintf(stderr, "%s: Cannot open 9x15 font\n",progname);
          exit( -1 );
     }
} /* load_font */


TooSmall(win, gc, font_info)
Window win;
```

```
GC gc;
XFontStruct *font_info;
{
     char *string1 = "Too Small";
     int y_offset, x_offset;

     y_offset = font_info->ascent + 2;
     x_offset = 2;

     /* output text, centered on each line */
     XDrawString(display, win, gc, x_offset, y_offset, string1,
                 strlen(string1));
} /* TooSmall */
```

```
/*\
 *    Filename: draw.c
 *
 *    Created:  March 15, 1992 - converted from bars.c
 *
 *    creator:  Carl Albrecht
 *    Modified: Robert Swenson - April 3, 1992 - added draw_data
 *    Modified: RAS - May 6 - reduced window/drawing size to fit on PC
 *
 *    Description:
 *              Draw in the window
\*/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdio.h>
#include "show.h"

extern Display *display;


#define NumStrings  30   /* number of strings to print on the screen */
#define   TextBarWidth   280  /* width in pixels of region containing text
*/
#define TextTab          9    /* # of pixels to tab over for uncentered */
#define VOffset          10   /* move down x pixels before typing */
#define StatBox          225  /* y Coord of the Cabin Status box */
#define StatBarTitles    420  /* "base" of status bars */
#define NumSegments 70   /* # of unconnected lines to draw */
#define WordsInPic  25   /* # of words to print into the picture */
#define BarGraphMaxHeight     100
#define   num_beds  4    /* num of names to switch when beds switch */


draw_text(win, gc, font_info, win_width, win_height, bed_flag)
Window win;
GC gc;
XFontStruct *font_info;
unsigned int win_width, win_height;
int  bed_flag;
{
     static char *strings[NumStrings] = {
                    "NASA/USRA",
                         "ADT",
                         "at",
                    "Kansas State University",
                    "Carl Albrecht   Roger Burjes",
                    "Dan Waldeck     Stan Kroeker",
```

```
                "Maury Wilmoth   Tim Sprecker",
                "Mike Honas",
                "Mike Brockway   Dr. Cogley",
                "Paul Snider     Dr. Gustafson",
                "Robert Swenson  Robert Young",

                "Cabin Status Panel",
                "Carbon    Tank     Fan",
                "Dioxide   Press    Speed",

                "Fan",    /*14*/          /* words in picture */
                "Dryer",
                "Pump",         /*16*/
                "",
                "",        /*18*/
                "Desorb",
                "Tank",         /*20*/
                "Pre",
                "Cooler", /*22*/
                "CO",
                "2",       /*24*/
                "Rehumidifier",
                "Absorb",       /*26*/
                "Storage"
                };
static struct coords {   /* to make drawing the picture easier */
                /* lets have an array of coordinates and */
                /* strings[] indexes and just run through */
                /* the array printing the words */
     int x,          /* coord to put the word at */
         y,
         the_word;  /* which strings[] to put at x,y */
     } the_coords[WordsInPic] = {
          {410,320,14},
          {560,100,16},
          {545,240,23},
          {565,245,24},
          {545,355,23},
          {565,360,24},
          {465,145,20},
          {470,335,21},
          {460,350,22},
          {470,105,23},
          {490,110,24},
          {455,125,27}
          },
      bed_coords[2][num_beds] = { { /* words to change when the */
                          /* beds switch [2] states */
          {320,240,15},
          {545,260,19},  /* Desorb */
          {545,375,26},  /* Adsorb */
          {295,450,25}
```

```
            },
            {
            {320,240,25},
            {545,260,26},
            {545,375,19},
            {295,450,15}
            }
            };
    int i;
    int len;
    int width;
    int font_height = font_info->ascent + font_info->descent;


   for (i=0;i<4;i++)    /* CENTER THE TITLES */
   {
    len = strlen(strings[i]);     /* len for XTextWidth and XDrawString */
    width=XTextWidth(font_info,strings[i],len);/* str width for centering
*/
    XDrawString(display, win, gc, (TextBarWidth - width)/2,
         VOffset + (i + 1) * (1 + font_height), strings[i], len);
   }

   for (i=4;i<11; i++)  /* SHOW THE NAMES */
   {
    len = strlen(strings[i]);     /* len for XTextWidth and XDrawString */
    XDrawString(display, win, gc, TextTab,
         VOffset + (i + 2) * (1 + font_height), strings[i], len);
   }

            /* SHOW CABIN STATUS TITLE */
    len = strlen(strings[11]);    /* len for XTextWidth and XDrawString */
    width=XTextWidth(font_info,strings[11],len);
    XDrawString(display, win, gc, (TextBarWidth - width)/2,
         StatBox + 2 * VOffset, strings[11], len);

   for (i=12;i<14;i++)  /* SHOW STATUS BAR TITLES */
   {
    len = strlen(strings[i]);     /* len for XTextWidth and XDrawString */
    XDrawString(display, win, gc, TextTab,
         StatBarTitles + (i - 11) * (1 + font_height),
         strings[i], len);
   }
   for(i=0;i<WordsInPic;i++) /* PUT WORDS IN THE PICTURE */
   {
    len = strlen(strings[the_coords[i].the_word]);
    width=XTextWidth(font_info,strings[the_coords[i].the_word],len);
    XDrawString(display, win, gc, the_coords[i].x,
         the_coords[i].y, strings[the_coords[i].the_word], len);
   }

   for(i=0;i<num_beds;i++)   /* PUT IN THE bed names */
```

```
    {
     len = strlen(strings[bed_coords[bed_flag][i].the_word]);
     width=XTextWidth(font_info,strings[bed_coords[bed_flag][i].the_word],
               len);

          /* INCASE the beds just switched, clear the area where this */
          /* word is about to go to avoid overwrite */
     /* XClearArea(); */


     XDrawString(display, win, gc, bed_coords[bed_flag][i].x,
               bed_coords[bed_flag][i].y,
               strings[bed_coords[bed_flag][i].the_word],len);
    }
} /* draw_text */
```

```
#define x_CO2  60   /* x coord of CO2 tank bar graph */
#define x_tank 120  /* x coord of CO2 tank pres bar graph */
#define   x_rpm     195  /* x coord of pump rpm bar graph */
#define x_des   322     /* x coord of Deisecant bed box in diagram */
#define x_fan   411 /* x coord of the fan */
#define x_pre  460  /* X coord of precooler */
#define x_CO2des 550     /* x coord of Zeolite beds */
#define BarWidth 10 /* width of status bars */
#define BoxWidth 47     /* width of the diagrams boxes */
#define C_Width      28  /* width of the fan circle */
#define C_Height 28 /* height of the fan circle */
#define Boxheight   53   /* height of boxes in pixels */
#define y_box   2
#define y_rpm  0
#define y_pres 1

draw_graphics(win, gc, window_width, window_height, code)
Window win;
GC gc;
unsigned int window_width, window_height;
struct diag_data *code;
{
     int x, y, oldy[3];
     int height;
     int  width;
     FILE *my_data;
     int i, j, Time_Step, rpm;
     float Pres;
     static XSegment     segments[NumSegments] = {
                    { 0, StatBox, TextBarWidth, StatBox},
                    { TextBarWidth, 0, TextBarWidth, 0},
                    { 0, StatBarTitles, TextBarWidth, StatBarTitles},
```

```
{ 280,190,320,190},  /* pipe A top */
{ 280,200,320,200}, /* A Bottom */
{ 280,400,320,400}, /* B Top */
{ 280,410,320,410}, /* B Bot */
{ 370,180,550,180}, /* G Top */
{ 370,190,550,190}, /* G Bot */
{ 370,200,400,200}, /* D Top */
{ 370,210,390,210}, /* D Bot */

{ 370,390,390,390}, /* C Top */
{ 370,400,400,400}, /* C Bot */
{ 370,410,550,410}, /* H Top */
{ 370,420,550,420}, /* H Bot */
{ 390,210,390,390}, /* E left */
{ 400,200,400,280}, /* E Right Top */
{ 400,400,400,290}, /* E Right Bot */
{ 400,280,410,280}, /* F Top */
{ 400,290,410,290}, /* F Bot */
{ 440,280,460,280}, /* I Top */
{ 440,290,460,290}, /* I Bot */
{ 510,280,520,280}, /* J Top */
{ 510,290,520,290}, /* J Bot */
{ 520,280,520,200}, /* K Top Left */
{ 520,290,520,400}, /* K Bot Left */
{ 520,200,550,200}, /* L Top */
{ 520,400,550,400}, /* M Bot */
{ 530,210,530,390}, /* K Right */
{ 530,210,550,210}, /* L Bot */
{ 530,390,550,390}, /* M Top */
{ 600,200,610,200}, /* N Bot */
{ 600,190,610,190}, /* N Top */
{ 600,410,620,410}, /* O Bot */
{ 600,400,610,400}, /* O Top */
{ 610,400,610,200}, /* P Bot Left */
{ 425,270,425,300}, /* Fan blades */
{ 410,285,440,285}, /* Fan Blades */
{ 416,275,435,295}, /* Fan Blades */
{ 416,295,435,275}, /* Fan Blades */
{ 610,190,610,80},  /* P Top Left */
{ 620,410,620,70},   /* P Right */
{ 610,80,580,80},    /* Q Bot */
{ 620,70,590,70},    /* Q Top */
{ 560,60,510,60},    /* R Bot */
{ 570,50,510,50},    /* R Top */
{ 450,60,370,60},    /* S Bot */
{ 450,50,370,50},    /* S Top */
{ 370,40,370,70},    /* Draws the octagon/OGA */
{ 300,40,300,70},    /* Draws the octagon/OGA */
{ 350,20,320,20},    /* Draws the octagon/OGA */
{ 350,90,320,90},    /* Draws the octagon/OGA */
{ 370,40,350,20},    /* Draws the octagon/OGA */
{ 370,70,350,90},    /* Draws the octagon/OGA */
```

```
                    { 300,40,320,20},     /* Draws the octagon/OGA */
                    { 300,70,320,90},     /* Draws the octagon/OGA */
                    { 565,55,570,60},     /*Draws the pumps*/
                    { 585,75,580,70},     /* Draws the pump */
                    { 460,250,480,280},/*Draws the PreCooler*/
                    { 480,280,485,270},
                    { 485,270,490,280},
                    { 460,330,480,300},
                    { 480,300,485,310},
                    { 485,310,490,300},
                    { 490,300,510,330},
                    { 490,280,510,250}
                };

      segments[1].y2 = window_height;     /* window_height is a var so it
can't
                                be used in the aggragrate assignment */

            /* draw the picture */
      XDrawSegments(display, win, gc, segments, NumSegments); /* draw pipes
*/
      XDrawRectangle(display,win,gc,x_des,170,47,Boxheight); /* AA */
      XDrawRectangle(display,win,gc,x_des,380,47,Boxheight); /* BB */
      XDrawRectangle(display,win,gc,x_pre,260,50,Boxheight); /* EE */
      XDrawRectangle(display,win,gc,x_CO2des,170,50,Boxheight);/* CC */
      XDrawRectangle(display,win,gc,x_CO2des,380,50,Boxheight);/* DD */

      /* This draws the fan */
      XDrawArc(display,win,gc,410,270,30,30,0,360 * 64);

      /* This draws the pump */
      XDrawArc(display,win,gc,560,50,30,30,0,360 * 64);

      /* This draws the center of the pump */
      XDrawArc(display,win,gc,570,60,10,10,0,360 * 64);

      /* This draws the CO2 Storage Tank */
      XDrawArc(display,win,gc,450,25,60,60,0,360 * 64);

         /* When drawing/redrawing the whole pic, draw complete level bars
*/
     /* instead of just the changed part */
         code->oldCO2 = 0;
         code->oldTank_Pres = 0;
         code->oldPump_Speed = 0;
         code->oldAbsorb = 0;
         code->oldDesorb = 0;
         code->oldCO2_Ab = 0;
         code->oldCO2_De = 0;

     draw_data(win,gc,width,height,code,y);
} /* draw_graphics */
```

```
draw_data(win,gc,width,height,code,y)
Window    win;
GC   gc;
int  width;
int  height;
struct diag_data *code;
int  y;
{
                /* draw the moving bars */
if (code->CO2 > BarGraphMaxHeight)
     code->CO2 = BarGraphMaxHeight;

         height = BarGraphMaxHeight - code->CO2;
         XClearArea(display, win, x_CO2, StatBarTitles - BarGraphMaxHeight,
               BarWidth, height, 0);

         y = StatBarTitles - code->CO2;
         height = code->CO2;
         XFillRectangle(display, win, gc, x_CO2, y, BarWidth, height);


 if (code->Tank_Pres > BarGraphMaxHeight)
      code->Tank_Pres = BarGraphMaxHeight;

          XClearArea(display, win, x_tank, StatBarTitles -
 BarGraphMaxHeight,
                BarWidth, BarGraphMaxHeight - code->Tank_Pres, 0);

          y = StatBarTitles - code->Tank_Pres;
          height = code->Tank_Pres;
          XFillRectangle(display, win, gc, x_tank, y, BarWidth, height);


 if (code->Pump_Speed > BarGraphMaxHeight)
      code->Pump_Speed = BarGraphMaxHeight;

          XClearArea(display, win, x_rpm, StatBarTitles - BarGraphMaxHeight,
                BarWidth, BarGraphMaxHeight - code->Pump_Speed, 0);

          y = StatBarTitles - code->Pump_Speed;
          height = code->Pump_Speed;
          XFillRectangle(display, win, gc, x_rpm, y, BarWidth, height);


                /* draw the box fillings */

       if (code->Absorb > code->oldAbsorb)
          {
```

```
    y = 222 - code->Absorb;
    height = code->Absorb- code->oldAbsorb;
    XFillRectangle(display, win, gc, x_des+2, y, BoxWidth, height);
}
else if (code->Absorb < code->oldAbsorb)
{
    y = 222 - code->oldAbsorb;
    height = code->oldAbsorb - code->Absorb;
    XClearArea(display, win, x_des+2, y, BoxWidth-2,height,0);
}

if (code->Desorb > code->oldDesorb)
   {
    y = 432 - code->Desorb;
    height = code->Desorb - code->oldDesorb;
    XFillRectangle(display, win, gc, x_des+2, y, BoxWidth, height);
}
else if (code->Desorb < code->oldDesorb)
{
    y = 432 - code->oldDesorb;
    height = code->oldDesorb - code->Desorb;
    XClearArea(display, win, x_des+2, y, BoxWidth-2, height,0);
}

if (code->CO2_Ab > code->oldCO2_Ab)
   {
    y = 222 - code->CO2_Ab;
    height = code->CO2_Ab - code->oldCO2_Ab;
    XFillRectangle(display, win, gc, x_CO2des+2, y, BoxWidth, height);
}
else if (code->CO2_Ab < code->oldCO2_Ab)
{
    y = 222 - code->oldCO2_Ab;
    height = code->oldCO2_Ab - code->CO2_Ab;
    XClearArea(display, win, x_CO2des+2, y, BoxWidth, height,0);
}

if (code->CO2_De > code->oldCO2_De)
   {
    y = 432 - code->CO2_De;
    height = code->CO2_De - code->oldCO2_De;
    XFillRectangle(display, win, gc, x_CO2des+2, y, BoxWidth, height);
}
else if (code->CO2_De < code->oldCO2_De)
{
    y = 432 - code->oldCO2_De;
    height = code->oldCO2_De - code->CO2_De;
    XClearArea(display, win, x_CO2des+2, y, BoxWidth, height,0);
}

        /* draw the fan moving */
```

```
    if (code->sys == 1)
    {
        XClearArea(display, win, x_fan, 271, C_Width, C_Height,0);
        XDrawArc(display,win,gc,410,270,30,30,0,360 * 64);
        XDrawLine(display,win,gc,425,270,425,300);
        XDrawLine(display,win,gc,410,285,440,285);
        XDrawLine(display,win,gc,416,275,435,295);
        XDrawLine(display,win,gc,416,295,435,275);
    }
} /* draw_data */
```